ESCUELA TÉCNICA SUPERIOR DE INGENIERIA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

PROYECTO FINAL DE CARRERA

Realización de un programa con interfaz de usuario para la generación de patrones de potencia arbitrarios con la fuente de alimentación MAGDRIVE

AUTOR: Pablo Bermúdez Alemán
DIRECTOR: Juan Monzó Cabrera
CODIRECTOR: Francisco Javier Clemente Fernández
Marzo 2012

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. Introducción.

Las microondas ocupan una región del espectro electromagnético (EM) que comprende un intervalo de frecuencias desde los 300 MHz hasta los 300 GHz, dentro del cual se pueden encontrar dos tipos de aplicaciones.

La primera son las aplicaciones para la transmisión de información, tales como, enlaces con satélites, radar, comunicaciones móviles, transmisión de radio, etcétera.

La segunda son las aplicaciones basadas en el uso de la energía de microondas, siendo este el caso en el que se va a enfocar este proyecto. En este tipo de aplicaciones, la onda no se modula e interacciona directamente con un sólido o un líquido, sino que la onda electromagnética transporta la energía que se transmite directamente al material en cuestión.

En la actualidad, el uso de aplicaciones industriales basadas en el uso de la energía de microondas forma parte del entorno que nos rodea, desde los hornos microondas que se encuentran en nuestras casas, hasta los hornos industriales que se usan para el secado de alimentos, secado de maderas, esterilización, secado de cerámicas o papel entre otras aplicaciones. Debido a este diverso uso de las aplicaciones industriales se ha planteado este proyecto final de carrera.

El objetivo planteado en este proyecto, es la creación de una interfaz en Matlab con la que se pueda controlar la fuente de alimentación Magdrive1000 para generar patrones arbitrarios de potencia con el objetivo de calentar objetos. Según se eleva la potencia transmitida, las temperaturas medidas en la cavidad van aumentando.

Se han generado tres tipos de interfaces. Una interfaz simple, en la que el usuario introduce distintos valores de potencia durante eligiendo el tiempo durante el cual se transmiten cada uno de ellos, otra que actúa como la simple pero que en esta ocasión se pueden transmitir las distintas potencias por dos puertos de comunicación para poder usar dos fuentes a la vez para transmitir distintos valores de potencia, y por último una interfaz en la que se transmiten los valores de potencia siguiendo funciones tales como: senoidal, diente de sierra, triangular y exponencial. Pudiendo seleccionar en cada caso, el número repeticiones del pulso y el tiempo de cada periodo y la potencia máxima transmitida.

Una vez creadas las distintas interfaces, se comprobó su funcionamiento en el laboratorio obteniendo una serie de resultados en función de la temperatura, los cuales se han analizado detenidamente para comprobar la eficiencia de cada una de las interfaces.

2. Fundamentos del calentamiento por microondas.

2.1 Breve historia de aplicaciones industriales del calentamiento por microondas.

La tecnología de las aplicaciones industriales de las microondas desde sus primeros usos durante la segunda guerra mundial hasta el día de hoy ha variado considerablemente.

Fue durante el curso de un proyecto de investigación relacionado con el radar, alrededor de 1946, que el doctor Percy Spencer, ingeniero de la Raytheon Corporation, descubrió que un dulce que tenía en su bolsa se había derretido cuando estaba probando un nuevo tubo al vacío llamado magnetrón. Tras la realización de diversas observaciones con alimentos se pudo comprobar que el calor producido por las microondas los calentaba.

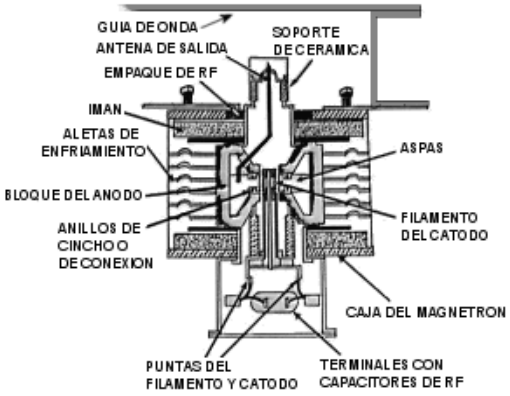

Figura 1 : Corte transversal de un magnetrón.

El doctor Spencer diseñó una caja metálica con una abertura en la que introdujo energía de microondas. Esta energía, dentro de la caja, no podía escapar y por lo tanto creaba un campo electromagnético de mayor densidad. Cuando se le colocaba alimento se producía energía de microondas y la temperatura del alimento aumentaba rápidamente, se le denominó horno microondas.

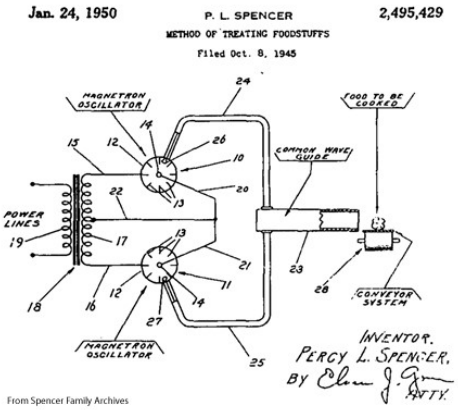

Figura 2: Imagen de la patente original del horno de microondas.

Durante este periodo la FCC [1] estaba estableciendo un procedimiento de asignación de frecuencias, momento en el cual GE [2] y Raytheon pidieron que se asignara una frecuencia para el horno de microondas. GE solicitó una frecuencia alrededor de los 915 MHz de la banda L/UHF mientras que Raytheon solicitó que se usara una frecuencia dentro de la banda S, actualmente se usa a 2.45 GHz.

Con el paso de los años se fue investigando e innovando desarrollándose avances, tales como, la creación del magnetrón de onda continua precursor de los actualmente usados, la invención de los agitadores de modos y filtros de las puertas usadas en los hornos de microondas.

A principios de los años sesenta, se implantaron microondas domésticos en casi todos los hogares hasta que las principales compañías buscaron otras alternativas a los hornos de microondas domésticos, surgiendo así los hornos de microondas industriales lo cual conllevó el crecimiento gradual de las aplicaciones industriales del calentamiento por microondas.

2.2 Conceptos básicos del calentamiento por microondas.

2.2.1 Introducción.

En este apartado se van a revisar los conceptos básicos del calentamiento por microondas. En primer lugar, se partirá de las ecuaciones de Maxwell para comprender el calentamiento volumétrico por microondas. A continuación se profundizará en el concepto de las ondas planas, que ayuda a la predicción de cómo se calentará un objeto dentro de un horno microondas. Una vez explicado el concepto de cómo se produce el calentamiento por microondas, se expondrá por último, cómo influye el agua y la humedad en aquellos materiales que se quieren calentar.

2.2.2 Calentamiento volumétrico por microondas.

Los campos electromagnéticos pueden propagarse a través de cualquier medio dieléctrico o magnético siguiendo las cuatro ecuaciones de Maxwell:

Ley de Faraday	$\nabla x \vec{E} = -\frac{\partial \vec{B}}{\partial t}$	(2.1)
Ley de Gauss para la electricidad	$\nabla \cdot \vec{D} = \rho$	(2.2)
Ley de Gauss para el magnetismo	$\nabla \cdot \vec{B} = 0$	(2.3)
Ley de Ampère – Maxwell	$\nabla x \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t}$	(2.4)

Donde:

$\vec{E}$ = Vector intensidad de campo eléctrico.

$\vec{D}$ = Desplazamiento eléctrico.

$\vec{B}$ = Vector inducción magnética.

$\vec{H}$ = Intensidad de campo magnético.

ρ = Densidad de carga volumétrica.

$\vec{J}$ = Densidad superficial de corriente.

Para la resolución de estas ecuaciones se usan las siguientes ecuaciones que relacionan los vectores intensidad e inducción a través de dos parámetros conocidos que son la permitividad eléctrica ε y la permeabilidad magnética μ .

$$\vec{D} = \varepsilon \vec{E} \quad (2.5)$$

$$\vec{B} = \mu\vec{H} \tag{2.6}$$

De la relación de estas ecuaciones se obtiene el vector de Poynting, la potencia absorbida por un material dieléctrico durante un proceso de calentamiento.

$$\vec{S} = \vec{E}\ x\ \vec{H} \tag{2.7}$$

Si se integra el vector de Poynting a lo largo de toda la superficie del cuerpo y se aplica el teorema de la divergencia, se obtiene la siguiente ecuación:

$$\int(\vec{E}\ x\ \vec{H})\cdot d\vec{S} = \int \nabla\cdot(\vec{E}\ x\ \vec{H}^*)\cdot d\vec{V} = -j\omega\int(\mu_0\mu'\vec{H}^*\cdot\vec{H} - \varepsilon_0\varepsilon\vec{E}\cdot\vec{E}^*)\,dV - \int\omega\varepsilon_0\varepsilon''\vec{E}\cdot\vec{E}^*dV \tag{2.8}$$

Donde:

ε' = Constante dieléctrica relativa.

ε'' = Factor de pérdidas.

μ'= La permeabilidad relativa del material dieléctrico.

ε_0 = Permitividad del vacío.

μ_0 = Permeabilidad del vacío.

Por otro lado la ecuación que define la potencia media es:

$$\vec{P} = -\frac{1}{2}\int Re(\vec{E}\ x\ \vec{H}^*)dS \tag{2.9}$$

Si se compara con la ecuación antes expuesta, se puede obtener la potencia absorbida y disipada por un cuerpo:

$$P = \frac{1}{2}2\pi\varepsilon_0\int\varepsilon''(V)\vec{E}\cdot\vec{E}^*\,dV \tag{2.10}$$

En este caso el término de calentamiento volumétrico generado por microondas se puede expresar como:

$$Q_{gen} = 2\pi f\varepsilon_o\varepsilon''\left|\vec{E}_{rms}\right|^2 \tag{2.11}$$

Donde:

Q_{gen} = Calor volumétrico generado por la energía de microondas (W/m^3).

f = Frecuencia (Hz).

$\vec{E}_{rms}$ = Campo eléctrico que viene dado según su valor eficaz.

En general la intensidad de campo eléctrico a lo largo del cuerpo no tiene porqué ser constante. Por lo que la generación de calor puede ser muy diferente a lo largo del cuerpo que se desea calentar en función del diseño del equipo (el horno).

Por lo tanto para obtener unos resultados más cercanos a la realidad, la formulación del calor volumétrico habrá que expresarlo en función del punto en el que es evaluado dentro del material y del instante temporal en el que se evalúa. El motivo por el que se debe evaluar dentro del instante temporal, es porque, cuando el material está siendo calentado, puede sufrir cambios físico-químicos que afecten a su estructura interna y por lo tanto a sus propiedades dieléctricas.

Por consiguiente si se expresa el calentamiento volumétrico como una función del espacio y el tiempo, la ecuación final sería:

$$Q_{gen}(x,y,z,t) = 2\pi f \varepsilon_o \varepsilon''(x,y,z,t)\left|\vec{E}_{rms}(x,y,z,t)\right|^2 \qquad (2.12)$$

Donde:

x, y, z, t = Representación de la situación cartesiana del punto evaluado y el tiempo.

De esta última ecuación se puede deducir que los principales factores que afectan al calentamiento del dieléctrico son: la frecuencia, el factor de pérdidas del material y la distribución espacial de campo eléctrico.

A continuación se explicarán cada uno de estos factores.

La frecuencia:

El calentamiento volumétrico, tal y como se puede apreciar es directamente proporcional a la frecuencia usada, es decir, si se usa una potencia radiada constantemente dentro de nuestro horno de microondas el calentamiento será mayor conforme se suba esta frecuencia.

Sin embargo se debe tener en cuenta que al subir la frecuencia la penetración de la energía electromagnética disminuye y que por lo tanto el calentamiento se vuelve más superficial. Por consiguiente cuando los materiales que se desean calentar tienen un grosor considerable, es conveniente usar un rango de frecuencias bajas para que éste se caliente uniformemente.

Una vez comprendido este efecto explicado se debe tener en cuenta que los sistemas de microondas deben restringirse al uso de aquellas frecuencias permitidas por los gobiernos para este tipo de equipos. Las frecuencias más usadas para uso industrial, científico y médico (ICM) son 2.45 GHz y 915 MHz.

Factor de pérdidas:

Indica la mayor o menor capacidad del material dieléctrico para absorber energía de microondas. De tal forma que cuanto mayor sea el factor de pérdidas de un material, mayor será la absorción de microondas.

También se debe tener en consideración que el factor de pérdidas no tiene porqué ser constante en todo el material, ya que existen materiales heterogéneos y cuya distribución de permitividad no es constante a lo largo del espacio.

Además dado que el cuerpo que sea calentado tendrá variaciones de temperatura y humedad o variaciones de estructuras químicas en su interior, el factor de pérdidas también podrá evolucionar con el tiempo durante el proceso de calentamiento por microondas.

Campo eléctrico:

En la fórmula se puede ver como el calentamiento por microondas es proporcional al cuadrado del módulo del campo eléctrico, lo cual implica que cuanto mayor sea el campo eléctrico en un punto del material, mayor será la subida de la temperatura.

Lo que interesaría, es que la distribución del campo eléctrico sea lo más uniforme posible en todo el material para que el calentamiento sea también uniforme. Cuando esto no sucede, partes del cuerpo se calientan mucho más que otras obteniendo 'hot spots' y 'cold spots'.

2.2.3 Aproximación a través de ondas planas.

Calcular el campo eléctrico de un material es fundamental para poder predecir cómo se calentará. Dicho cálculo se puede realizar mediante métodos numéricos complejos y determinadas herramientas como por ejemplo la herramienta PDETOOL de Matlab.

En este apartado se van a explicar brevemente las ondas planas, lo cual da una idea bastante clara e ilustra de forma bastante sencilla lo que sucede en un material con pérdidas dieléctricas cuando un campo incide sobre él.

Propagación de ondas planas en dieléctricos sin pérdidas.

Partiendo de la ecuación de maxwell ya explicada en el apartado 2.2.2, se puede llegar a la ecuación de Helmholtz, que es la ecuación de una onda suponiendo variación senoidal a la frecuencia angular ω

$$\nabla^2 \vec{E} + k^2 \vec{E} = 0 \tag{2.13}$$

k = Número de onda del medio de propagación considerado

Su resolución por separación de variables da como resultado el campo eléctrico para cada una de las componentes:

$$E_x = A_x e^{-jk_x x} + B_x e^{jk_x x}$$
$$E_y = A_y e^{-jk_y y} + B_y e^{jk_y y} \tag{2.14}$$
$$E_z = A_z e^{-jk_z z} + B_z e^{jk_z z}$$

Como se debe cumplir que la divergencia de E sea nula, k y E serán perpendiculares siempre. También se cumplirá que:

$$k_x^2, k_y^2, k_z^2 > 0 \;\; y \;\; k^2 = k_x^2 + k_y^2 + k_z^2 \tag{2.15}$$

Las ondas planas presentan dos soluciones posibles para cada sentido en la dirección e propagación k:

$$E = E_{o+} e^{-jkrr} + E_{o-} e^{jkrr} \tag{2.16}$$

representan dos ondas planas propagándose en la dirección k y -k respectivamente y con fase constante.

La distancia mínima entre planos de cada fase viene dada por la denominada longitud de onda ($\boldsymbol{\lambda}$):

$$\lambda = \frac{2\pi}{k} = \frac{2\pi}{\omega\sqrt{\mu\varepsilon}} = \frac{1}{f\sqrt{\mu\varepsilon}} \tag{2.17}$$

Donde:

ω = Frecuencia angular de la onda.

μ = Permeabilidad magnética del medio.

ε = Permitividad eléctrica.

Dado que son medios sin pérdidas tanto μ como ε serán reales, sin parte compleja.

De esta expresión obtenemos la velocidad de fase:

$$vp = \frac{1}{\sqrt{\mu\varepsilon}} \tag{2.18}$$

La velocidad de fase, marca la velocidad a la que viajan los frentes de onda planos. $vp = 3 \cdot 10^8 \; m/s$ en el vacío, pero para otros dieléctricos y medios de transmisión, la velocidad de propagación será menor.

En cuanto a la potencia media de una onda plana en el dieléctrico según la dirección de propagación se puede obtener según la ecuación:

$$\overrightarrow{P_m} = \frac{1}{2}\frac{E^2}{\eta}\,\hat{k} \tag{2.19}$$

Donde:

$\eta = \sqrt{\mu/\varepsilon}$ Es la impedancia de onda del medio.

Propagación de ondas planas en dieléctricos con pérdidas.

En este apartado ahora ε en lugar de ser una constante real la consideramos compleja $\varepsilon^* = \varepsilon' - j\varepsilon''$.

Las soluciones que obtenemos para la onda plana son exactamente las mismas aunque en este caso la constante de propagación k^*será compleja. Sin embargo, esto va a forzar un cambio en el comportamiento físico de la onda plana.

Si consideramos un material con una permitividad compleja $\varepsilon^* = \varepsilon' - j\varepsilon''$ y onda plana que se propaga en el sentido positivo del eje 'z' $\vec{E} = \overrightarrow{E_o} e^{-jkz}$, la constante de propagación obtenida será compleja.

$$k = \omega\sqrt{\mu\varepsilon} = \omega\sqrt{\mu\,(\varepsilon'' - \varepsilon'')} = \beta - j\alpha \qquad (2.20)$$

Donde:

α = Constante de atenuación.

β = Constante de propagación.

ε' = Constante dieléctrica.

ε'' = Factor de pérdidas.

Con lo cual el campo eléctrico instantáneo puede escribirse como:

$$\vec{E}(r,t) = Re\left[\vec{E}\cdot e^{j\omega t}\right] = Re\left[\overrightarrow{E_o}\, e^{-\alpha z}\, e^{-\beta z}\, e^{j\omega t}\right] = E_o e^{-\alpha z}\cos(\omega t - \beta t + \emptyset i) \qquad (2.21)$$

En la figura siguiente se muestra la evolución espacial del campo para diferentes instantes de tiempo. Puede observarse que la amplitud del campo decrece exponencialmente a medida en que avanzamos en la dirección de propagación (en este caso z).

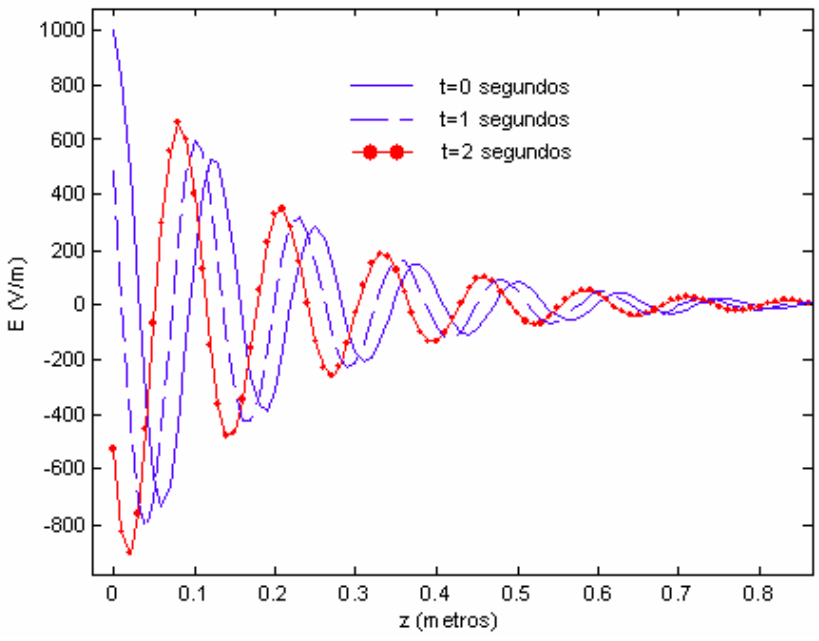

Figura 3 : Evolución campo eléctrico (I)

Las constantes de propagación y de atenuación se definen en este caso como:

$$\alpha = \omega\sqrt{\mu\varepsilon'}\left[\frac{\sqrt{1+\left(\frac{\varepsilon''}{\varepsilon'}\right)^2}-1}{2}\right]^{1/2} \tag{2.22}$$

$$\beta = \omega\sqrt{\mu\varepsilon'}\left[\frac{1+\sqrt{1+\left(\frac{\varepsilon''}{\varepsilon'}\right)^2}}{2}\right]^{1/2} \tag{2.23}$$

Donde la relación entre el factor de pérdidas y la constante dieléctrica se la denomina tangente de pérdidas:

$$tg(\delta) = \left(\frac{\varepsilon''}{\varepsilon'}\right) \tag{2.24}$$

De estas ecuaciones se pueden deducir las siguientes cuestiones:

1. La atenuación aumenta proporcionalmente con la frecuencia.
2. La atenuación aumenta proporcionalmente con la constante dieléctrica y con el factor de pérdidas.
3. La atenuación es proporcional a la permeabilidad del material.

Por lo tanto como se ha podido comprobar, la energía perdida por la onda plana es transformada en el cuerpo en forma de calor. Es decir, a medida que la onda se propaga cede energía al cuerpo, la cual se transforma en calor, tal y como marca la expresión (2.12). Así, según este criterio las zonas interiores del cuerpo recibirán menos calor que las exteriores.

Otra cuestión que se debe tener en cuenta es lo que ocurre cuando las pérdidas dieléctricas aumentan en el cuerpo. En la figura se muestra qué ocurre para diferentes constantes de atenuación:

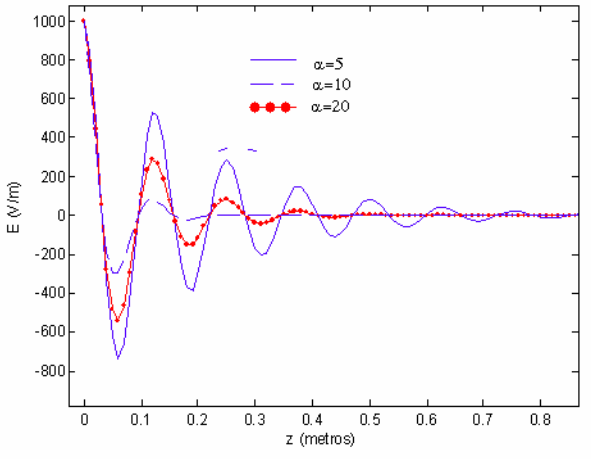

Figura 4 : Evolución campo eléctrico (II)

Se puede apreciar que a medida que α aumenta, la atenuación de la onda es mayor y por lo tanto, las zonas interiores reciben menos campo eléctrico, es decir, conforme aumentan las pérdidas dieléctricas, la diferencia de temperatura entre el interior y el exterior del cuerpo es mayor.

También se puede observar el efecto del aumento de la constante de propagación β (por el aumento de la frecuencia, de la constante dieléctrica, de la permeabilidad, o de la tangente de pérdidas). En la siguiente figura se muestra lo que ocurre para diferentes constantes de propagación.

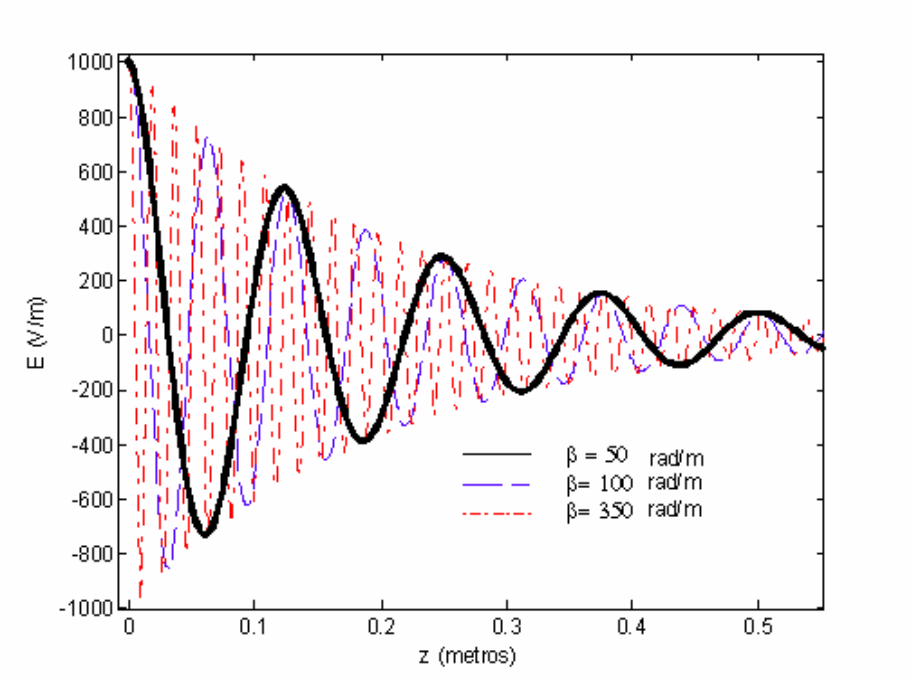

Figura 5 : Evolución campo eléctrico (III)

Se puede apreciar que a medida que β aumenta, las oscilaciones del campo eléctrico dentro del cuerpo también lo hacen. Sin embargo, las pérdidas de la onda plana se mantienen iguales en todos los casos.

Es decir, β, tiene relación con la longitud de onda dentro del material, pero no influye en la pérdida de potencia.

La ley de Lambert:

Método para hallar el calor volumétrico por microondas (Q_{gen} (W/m^3) e indica que el campo decae exponencialmente. Por tanto se puede establecer un flujo de potencia en la dirección de propagación (z) dado por:

$$\vec{P} = P_{superficie} e^{-2\alpha z} \hat{z} \qquad (2.25)$$

Donde:

$P_{superficie}$ = Potencia existente en la superficie del cuerpo.

También se debe destacar el concepto de profundidad de penetración, definida como aquella distancia desde la superficie a la cual la potencia decae hasta $1/e$:

$$D_p = \frac{1}{2\alpha} \tag{2.26}$$

En la siguiente figura se puede ver cómo el decaimiento del campo eléctrico a lo largo del cuerpo sigue una trayectoria exponencial:

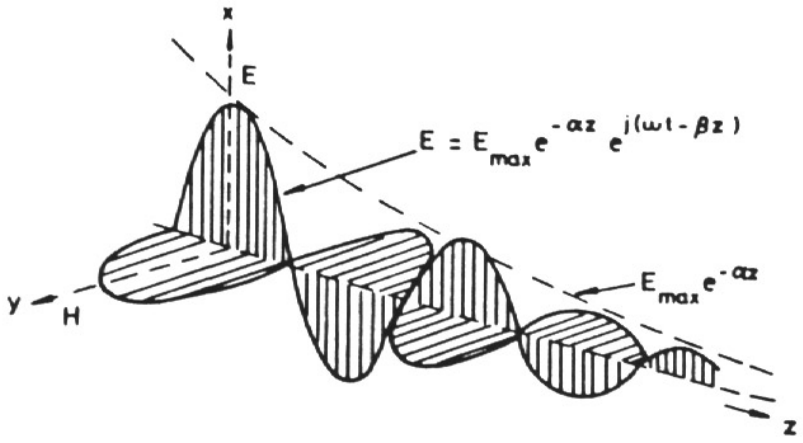

Figura 6: Campo electromagnético.

Para cuantificar el término de calor generado por absorción de microondas, según esta ley, se realiza un balance de la potencia absorbida en todo el material. Para ello se considerará que la potencia absorbida por unidad de superficie se puede expresar como:

$$P_s'' = \frac{P_{superficie}}{2(L_1L_2+L_2L_3+L_3L_1)} \tag{2.27}$$

Donde:

L_1, L_2, L_3 = Dimensiones de la siguiente muestra.

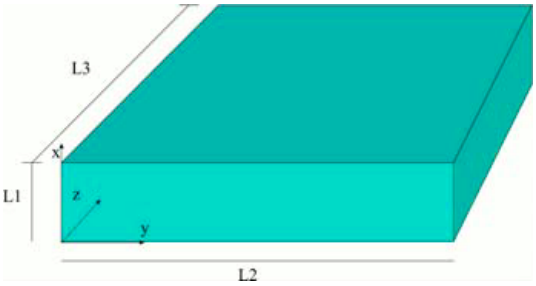

Figura 7: Muestra rectangular.

La potencia de microondas por unidad de área que se propaga en la dirección x desde la superficie será, según la ley de Lambert:

$$P_x'' = P_s'' e^{-2\alpha x} \tag{2.28}$$

Si ahora se deriva la potencia superficial respecto de **x**, y se tienen en cuenta las aportaciones de las dos caras de la muestra, obtendremos la potencia volumétrica absorbida en esta dirección:

$$Q_{gen} = P_x''' = 2\alpha \cdot P_s''[e^{2\alpha x} + e^{-2\alpha(L_1 - x)}] \tag{2.29}$$

Esta ley ha sido utilizada en muchos trabajos cuando se estudian procesos de secado en hornos microondas multimodo. Sin embargo, las condiciones sobre las que se pueden aplicar (condición de onda plana incidente sobre el dieléctrico) no se cumplen. A pesar de esto se utiliza debido, fundamentalmente, a su sencillez y al hecho de que para dieléctricos con altas pérdidas da unos resultados bastante aproximados al comportamiento experimental observado. No obstante, algunos autores desaconsejan su uso indiscriminado sobre todo para dieléctricos con bajas pérdidas.

2.2.4. Características de la materia (permitividad y permeabilidad).

Las pérdidas dieléctricas (ε*= ε'- jε'').

En los puntos anteriores se ha visto, desde un punto de vista teórico, por qué se produce el calentamiento asistido por microondas. Se han caracterizado los dieléctricos con pérdidas mediante una parte real (constante dieléctrica) y otra imaginaria (factor de pérdidas).

Sin embargo, no se han descrito los mecanismos físicos por los que producen las pérdidas dieléctricas, en las cuales nos centraremos. Básicamente las pérdidas dieléctricas se pueden explicar a partir de dos efectos que, a menudo, se producen simultáneamente, en mayor o menor grado: el efecto de polarización y el de conducción eléctrica.

El calentamiento por microondas consiste en la mayor o menor habilidad del dieléctrico para polarizar sus cargas o mover sus iones positivos y/o negativos a lo largo de su volumen. Este movimiento se ve forzado por el campo eléctrico externo. La imposibilidad de las moléculas polares o los iones para seguir los cambios rápidos del campo eléctrico es lo que provoca la disipación de energía en forma de calor.

La interacción de un campo eléctrico con un dieléctrico tiene su origen en la respuesta de las partículas con carga frente a dicho campo; las partículas se moverán a partir de su posición de equilibrio.

Existen dos mecanismos de polarización básicos:

1. Electrónico (movimiento de los electrones alrededor de su núcleo).
2. Molecular (distribuciones de carga en los átomos y moléculas no uniforme).

El mecanismo de polarización a su vez puede tener tres orígenes:

1. Polarización inducida debida al desplazamiento del núcleo frente al electrón.
2. Polarización permanente que se produce porque los dieléctricos contienen moléculas polares con una distribución de carga asimétrica (agua).
3. Polarización en las interfaces de diferentes dieléctricos en contacto (carga superficial o mecanismo de Maxwell-Wagner).

En la figura siguiente se muestran los mecanismos de la redistribución de iones cargados no negativamente y la reorientación de los dipolos bajo el efecto de un campo eléctrico externo. En campos alternos de gran frecuencia las moléculas e iones vibrarán o rotaran alrededor de una posición lo que hará que fricciones entre sí.

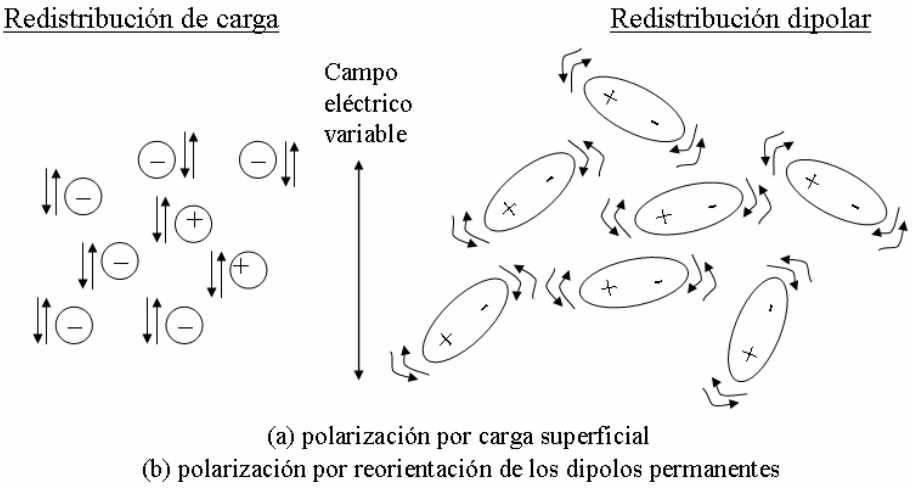

Figura 8: Mecanismos de Polarización.

Debido a que con la mayoría de las técnicas de medida resulta difícil separar las pérdidas debido a conducción y a la polarización, se suele expresar el factor de pérdidas definiendo un factor de pérdidas efectivo. Así, el factor de pérdidas se puede expresar como una contribución de ambos mecanismos (conducción y polarización):

$$\varepsilon''_{eff}(\omega) = \varepsilon''_d(\omega) + \varepsilon''_e(\omega) + \varepsilon''_a(\omega) + \varepsilon''_{MW}(\omega) + \left[\frac{\sigma}{\omega\varepsilon_o}\right] = \varepsilon''(\omega) + \left[\frac{\sigma}{\omega\varepsilon_o}\right] \qquad (2.30)$$

Donde:

d, e, a, MW = Pérdidas dieléctricas debido a los mecanismos dipolares, electrónicos, atómicos y el efecto Maxwell-Wagner.

σ = Conductividad del medio dieléctrico y agrupa las pérdidas por conducción.

El factor de pérdidas presenta una dependencia con la frecuencia, dado que a ciertas frecuencias dominan más unos fenómenos físicos sobre los otros. Los mecanismos más importantes desde el punto de vista de calentamiento por microondas serán, básicamente, los de polarización (que dependerán de la cantidad de agua o material polar que componga el material) y los de conductividad en continua.

Los demás mecanismos tienen mayor preponderancia a frecuencias cercanas al espectro infrarrojo y visible.

Las pérdidas dipolares: ecuaciones de Debye.

El modelo Debye está basado en la respuesta dieléctrica de relajación de una población ideal de dipolos que no interactúan unos con otros cuando están sometidos a un campo alterno externo.

La caracterización frecuencial de la rotación de las moléculas dipolares se lleva a cabo a partir de la rotación de un dipolo esférico que está en un medio viscoso. Así Debye llegó a la conclusión para electrolitos de que la dependencia de la permitividad con lal frecuencia seguía la forma:

$$\varepsilon^*(\omega) = \varepsilon_\infty + \frac{\varepsilon_S - \varepsilon_\infty}{1 + j\omega\tau} \tag{2.31}$$

Donde:

$\varepsilon_S, \varepsilon_\infty$ = Constantes dieléctricas a corriente continua y a muy alta frecuencia.

T = Tiempo de relajación del sistema, que controla el mecanismo de polarización.

Separando las partes real e imaginaria para la permitividad se obtiene:

$$\varepsilon'(\omega) = \varepsilon_\infty + \frac{\varepsilon_S - \varepsilon_\infty}{1 + \omega^2\tau^2} \tag{2.32}$$

$$\varepsilon''(\omega) = \frac{(\varepsilon_S - \varepsilon_\infty)\omega\tau}{1 + \omega^2\tau^2} \tag{2.33}$$

La interpretación de estas fórmulas indica que a baja frecuencia los dipolos tienen bastante tiempo para seguir las variaciones del campo eléctrico, por lo que el factor de pérdidas es muy pequeño. Por otra parte la constante dieléctrica toma su máximo valor porque la carga ligada toma su máximo valor y toda la energía consigue almacenarse.

Cuando la frecuencia aumenta los dipolos son incapaces de seguir los cambios del campo eléctrico por lo que no pueden volver a su posición original. Se llega a alcanzar un punto en el que el dipolo no puede realinearse contribuyendo menos a la polarización total. Esto hace que el material disipe energía en forma de calor.

Las pérdidas dipolares: ecuaciones de Maxwell-Wagner.

Las pérdidas debido al efecto interfacial o de Maxwell-Wagner son muy importantes en dieléctricos heterogéneos, lo cual se da en muchos cuerpos con agua y una matriz

sólida (frutas, cuero, madera). De igual manera al anterior tratamiento, existen formulaciones que caracterizan este tipo de polarización en función de la frecuencia.

Para dos dieléctricos con grosores d1 y d2, la permitividad compleja se puede escribir como:

$$\varepsilon^*_{MW}(\omega) = \varepsilon_\infty + \frac{\varepsilon_s - \varepsilon_\infty}{1+j\omega\tau} - j\frac{\sigma}{\omega\varepsilon_o} \quad (2.34)$$

La parte real de este modelo es la misma que la del modelo de Debye, sin embargo, el término de pérdidas incluye la conductividad (término en continua) σ. De esta forma el factor de pérdidas aumenta al disminuir la frecuencia debido a este término de conductividad.

Como conclusión a todo lo comentado se debe tener en cuenta que, cuanto más dipolar sea un material, más pérdidas tendrá debido a los dos efectos anteriores y por lo tanto más calor se generará en su interior.

Uno de los materiales con mayor comportamiento dipolar es el agua con lo que, aquellos materiales que contengan gran cantidad de agua se calentarán al aplicarles un campo a las frecuencias de microondas. A estas frecuencias el mecanismo que predomina en las pérdidas dieléctricas es el dipolar.

A menores frecuencias (en torno a 27.12 MHz, aplicaciones de calentamiento por radiofrecuencia) el fenómeno de calentamiento de agua ya no es dipolar (como se ha indicado antes), sino que ésta se calienta por conducción eléctrica debido a los iones positivos y negativos que el agua no pura presenta, como son el sodio, el cloro, etc.

Permitividad.

El conocimiento de las propiedades dieléctricas de los materiales que van a procesarse en un horno de microondas es fundamental por dos motivos:

− Dichas propiedades marcan cómo se propaga la energía en el cuerpo, determinando la longitud de onda, impedancia, velocidad y constante de propagación.

− También marcan cómo se transforma la energía de microondas en calor en el interior del dieléctrico tal y como indica la ecuación (2.11).

De hecho, un alto factor de pérdidas indica que el material es susceptible de ser tratado con microondas mientras que un factor de pérdidas reducido indica lo contrario, es decir, la permitividad eléctrica indica si un material es susceptible de ser tratado o no con microondas.

De aquí nace la necesidad de caracterizar las propiedades dieléctricas de los materiales en el contexto de la aplicación de las microondas a efectos de calentamiento.

Dependencia con el contenido de humedad.

Muchas aplicaciones del calentamiento por microondas implican eliminación de humedad del interior de un material o carga. Por tanto, la caracterización de la permitividad en función del contenido de humedad resulta fundamental para el diseño de los dispositivos de secado asistido por microondas.

Para caracterizar la cantidad de agua presente en un cuerpo frente a su masa seca (sin agua) se utiliza el contenido de humedad en base seca:

$$X = \frac{m - m_s}{m_s} \tag{2.35}$$

Donde:

X = Contenido de humedad en base seca.
m = La masa total del cuerpo en un instante determinado del proceso de deshidratación.
m_s = la masa de la matriz sólida del mismo cuerpo.

Se suele distinguir entre dos tipos de agua en el interior de los materiales:

1. Agua ligada, en contacto con las paredes de la estructura sólida, tiene un movimiento reducido, y
2. Agua libre, que se comporta como agua normal y por tanto tiene comportamiento polar.

En la figura se muestra la variación del factor de pérdidas (ε'') en función del contenido de humedad (X) de un sólido típico. Se pueden diferenciar claramente dos etapas con diferentes pendientes ligadas a los dos tipos de agua en el interior del cuerpo.

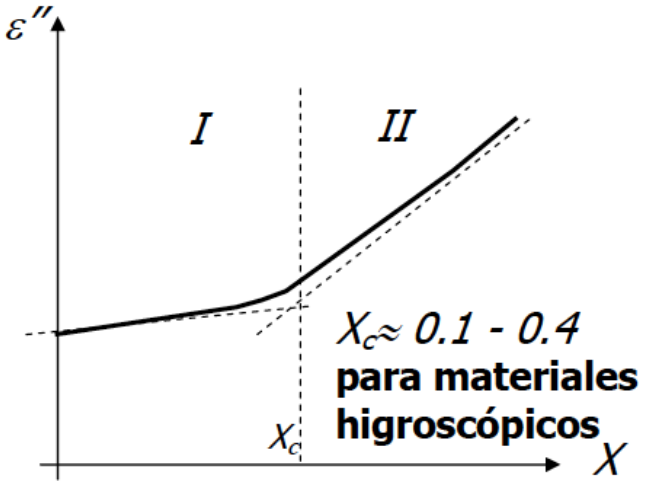

Figura 9: Permitividad y Factor de pérdidas ante la humedad.

La primera parte (con menor pendiente) es debida al agua ligada mientras que la mayor pendiente se observa para contenidos de humedad mayores en los cuales el

agua en el cuerpo es fundamentalmente libre. El cambio de pendiente se produce en el denominado contenido de humedad crítico.

La diferencia de comportamientos se debe a que el agua ligada tiene una menor movilidad y por lo tanto no rota tanto como la libre (menor carácter dipolar al estar ligada a la superficie del cuerpo por capilaridad y otro tipo de enlaces).

En la mayoría de las ocasiones se puede representar tanto el factor de pérdidas como la constante dieléctrica con una relación cuadrática:

$$\varepsilon'' = \varepsilon_o'' + AX + BX^2 \tag{2.36}$$

Donde:

ε_o, A, B = Constantes que se ajustan al comportamiento experimental del medio.

La pendiente de la curva en la zona II es muy importante para las aplicaciones de secado selectivo en las cuales el diferente comportamiento entre la zona I y la II es fundamental para que las partes con mayor humedad absorban mayor energía y por lo tanto se sequen a una mayor velocidad.

El secado selectivo no es tan eficiente para la zona I puesto que las diferencias en la absorción ($\boldsymbol{\varepsilon}''$) son mínimas para diferentes porcentajes de humedad.

Dependencia con la temperatura

Se han realizado multitud de estudios sobre la dependencia de la permitividad con la temperatura en alimentos, agua, madera, etc. De hecho está demostrado que el hielo presenta una permitividad mucho menor que el agua líquida. En la figura se muestra $\boldsymbol{\varepsilon}'$ para el Nylon a diferentes temperaturas para una frecuencia de 3GHz. Puede observarse cómo al aumentar la temperatura aumenta también el factor de pérdidas y su constante dieléctrica.

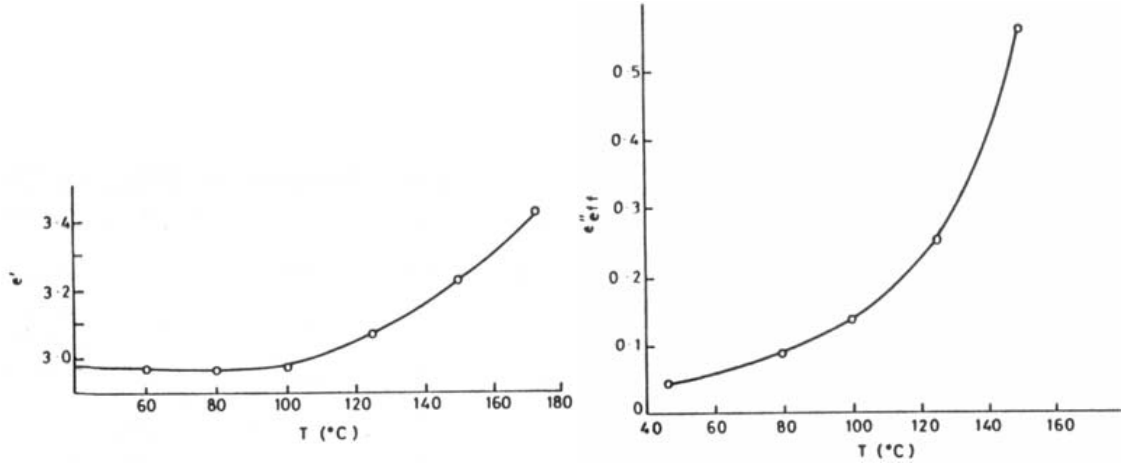

Figura 10: Propiedades dieléctricas Nylon.

Para cada material existe una temperatura crítica a la cual el material experimenta un cambio en sus propiedades dieléctricas muy acusado, tal y como ocurría con la dependencia con la humedad.

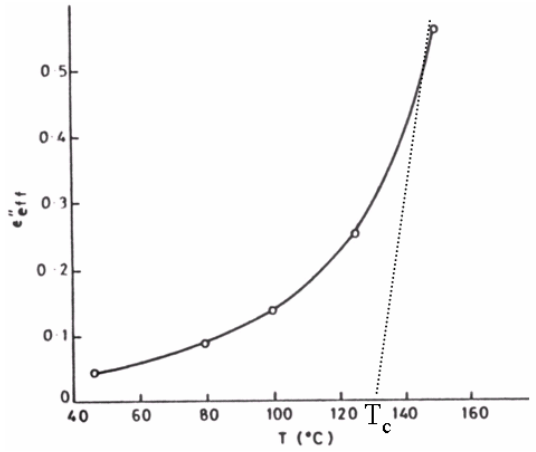

Figura 11: Temperatura crítica nylon (3 GHz).

La Avalancha De temperatura ('Thermal Runaway').

Si nos fijamos en la ecuación (2.11) y en las propiedades dieléctricas del Nylon presentadas anteriormente se puede llegar a que se produzca un incremento incontrolado de la temperatura denominado avalancha de temperatura.

Este hecho se produce porque el factor de pérdidas crece con la temperatura, con lo cual a mayor temperatura mayor absorción de potencia, lo cual redunda en un crecimiento de temperatura mayor.

Este crecimiento de la temperatura no controlado produce, o puede llegar a producir, daños en el material porque ciertos puntos de éste pueden llegar a calentarse tanto que el material no soporte la temperatura, mientras que otros puntos del material permanecen a mucha menor temperatura. Este efecto también se produce cuando se trata de descongelar productos, dado que aquellas zonas que se descongelan primero absorberán mucha más potencia que las otras, lo cual no asegura un descongelado uniforme.

2.2.5. Interacción microondas-materia.

En este apartado se va a explicar la ecuación del calor, la cual permite calcular la distribución de temperaturas en el interior de un material, una vez conocidos ciertos parámetros térmicos del mismo. Pero antes se explicarán brevemente una serie de mecanismos físicos que permiten que el calor se transfiera de un cuerpo a otro o se redistribuya en su interior.

Cuando a un material se le aplican microondas pueden darse tres fenómenos simultáneos y relacionados entre sí:

1. La transmisión de energía al sólido generada en su interior lo cual genera el calor necesario para elevar la temperatura del material (2.11).
2. El calor generado también puede utilizarse en el cuerpo para evaporar la humedad que éste contenga. Por lo que se puede producir una transferencia de humedad, en estado líquido o gaseoso, desde el interior del material hasta la superficie del mismo, lugar donde pasa a formar parte del medio externo.

3. El desarrollo de una energía mecánica, debida a los flujos de las fases líquidas y a las contracciones, dilataciones y distorsiones de la estructura del material sólido considerado.

Es decir, cuando se aplican microondas se está aportando una energía a un material que anteriormente estaba en equilibrio termodinámico. Esta energía cambiará el estado termodinámico del cuerpo bien incrementando su temperatura, bien evaporando el agua que el cuerpo contuviera.

Transferencia de calor convencional: Mecanismos Físicos.

1. La conducción del calor: conductividad térmica.

La transferencia de energía mediante conducción sucede a escala molecular o atómica, los átomos con mayor energía transfieren parte de la misma por colisión con los átomos caracterizados con una menor energía. Así, el calor siempre se propagará desde las zonas de mayor temperatura a las de menor temperatura.

Para modelar materiales heterogéneos con diferentes componentes (humedad, matriz sólida, etc.) se han propuesto muchos modelos, que intentan obtener una conductividad efectiva a partir de las distintas conductividades de las materias que componen el sólido húmedo. Uno de estos modelos es el de Maxwell cuya expresión es:

$$K_T = \frac{K_2[K_1+2K_2-2(1-e)(K_2-K_1)]}{[K_1+2K_2+(1-e)(K_2-K_1)]} \qquad (2.37)$$

Donde:

Ki = Conductividad de la fase i.

e = es la fracción de la fase 2 en el total.

2. La transferencia de calor por convección.

La convección tiene lugar cuando existe un fluido en movimiento sobre la superficie de un material, estando ambos a diferentes temperaturas. En este caso existen dos mecanismos de transferencia de energía que contribuyen al efecto global de la convección.

1. Transferencia de energía debida al movimiento aleatorio de las moléculas en la interfaz del fluido y el sólido, las cuales al chocar se transmiten energía de un modo parecido al proceso de conducción debido al gradiente de temperaturas que aparece entre los dos medios.
2. El fluido, con una velocidad dada, puede transferir parte de esta energía a la interfaz del sólido de forma macroscópica, ya que grandes agregados de moléculas se mueven dentro del fluido.

Por lo tanto, debido a estos mecanismos, la convección es un fenómeno superficial.

3. Fenómeno de radiación térmica.

La radiación térmica es energía emitida por un cuerpo que se encuentra a una determinada temperatura. Esta emisión de energía se produce incluso en el vacío. Esta radiación se atribuye a los cambios de configuración en los electrones y es transportada por medio de ondas electromagnéticas (las cuales pueden propagarse más efectivamente en el vacío que en cualquier otro medio).

Normalmente no se considera relevante este mecanismo de transmisión de calor frente al de convección puesto que su aportación al flujo total de calor suele ser muy pequeña comparada con los mecanismos de convección.

4. El calentamiento por microondas.

Desde hace muchos años se conoce la posibilidad de calentar materiales mediante la aplicación de ondas electromagnéticas en la banda de microondas. El origen de este calentamiento nace de la interacción del campo eléctrico con las moléculas o partículas que tienen una distribución de carga no nula, bien desplazándolas de lugar (fenómeno de conducción), bien haciéndolas girar alrededor de su posición de equilibrio (fenómeno de polarización).

Cuando el campo eléctrico cambia de dirección muy rápidamente, las partículas y moléculas intentan seguir estos cambios, bien desplazándose de un lado a otro del material, bien girando sobre sí mismas. Estos movimientos provocan fricciones que generan calor en el interior del material.

La ecuación 2.12, aquí repetida, muestra el calentamiento que se produce en un punto del cuerpo (calor volumétrico) en función de la frecuencia y del campo eléctrico y el factor de pérdidas en dicho punto.

$$Q_{gen}(x,y,z,t) = 2\pi f \varepsilon_o \varepsilon''(x,y,z,t)\left|\vec{E}_{rms}(x,y,z,t)\right|^2 \qquad (2.38)$$

LA ECUACIÓN DEL CALOR: Modelo para la predicción del calentamiento.

En la siguiente ecuación muestra un modelo que predice la temperatura de un cuerpo expuesto a microondas:

$$\rho \cdot c_p \frac{dT}{dt} = K_T \nabla^2 T + Q_{gen} \qquad (2.39)$$

Donde:

$KT\ (W/m^{\circ}C)$ = Conductividad térmica del material.

$\rho\ (kg/m^3)$ = Densidad.

$c_p (J/kg\ ^{\circ}C)$ = Calor específico.

$T(^{\circ}C)$ = Temperatura

$Qgen\ (W/m^3)$ = Calor generado por microondas según (2.12).

Esta ecuación es denominada ecuación del calor con el término de generación por microondas y es válida cuando en el cuerpo se produce un calentamiento por microondas y no suceden otros procesos tales como evaporación en su interior.

En la ecuación del calor, el término de la izquierda marca el incremento de temperatura en el cuerpo (el incremento de T será menor cuanto mayores sean ρ y c_p). El primer término de la parte derecha de la igualdad marca cómo se redistribuye el calor debido al fenómeno de conducción térmica, siendo $\nabla^2 T$ el operador laplaciana en cartesianas

$$\nabla^2 T = \frac{d^2T}{dx^2} + \frac{d^2T}{dy^2} + \frac{d^2T}{dz^2} \tag{2.40}$$

Si el material contiene humedad en su interior y ésta se evapora, la ecuación de calentamiento tiene un nuevo término que tiene en cuenta el calor perdido por dicha evaporación:

$$\frac{dT}{dt} = \alpha_T \nabla^2 T + \frac{e_v}{c_p} \Delta H_{ev} \frac{dX_1}{dt} + \frac{Q_{gen}}{\rho C_p} \tag{2.41}$$

Donde:

e_v= Coeficiente de evaporación interna (fracción de agua evaporada en el interior del material frente a la evaporada en su superficie). Sólo podrá tomar valores entre 0 y 1.

ΔH_{ev}= Vapor latente de evaporación (energía necesaria para evaporar un kilogramo de agua).

X_1 = Contenido de humedad líquido. Disminuye a lo largo del tiempo, por lo tanto su derivada será negativa por lo que tenderá a disminuir la temperatura global del cuerpo.

Debido a esto cundo un cuerpo posee agua, ésta absorbe la energía para evaporarse, no pudiendo usarse para elevar la temperatura del cuerpo hasta que se ha evaporado el agua.

2.3 Componentes y sistemas de un horno de calentamiento por microondas.

2.3.1 Introducción.

En este apartado se van a explicar los sistemas de calentamiento por microondas. Tratando en primer lugar la arquitectura general de los hornos industriales para a continuación centrarnos en la figura del magnetrón y sus distintas fuentes de alimentación.

2.3.2 Esquema básico de calentamiento por microondas.

En la siguiente figura se muestra el diagrama básico que cualquiera sistema de microondas presenta.

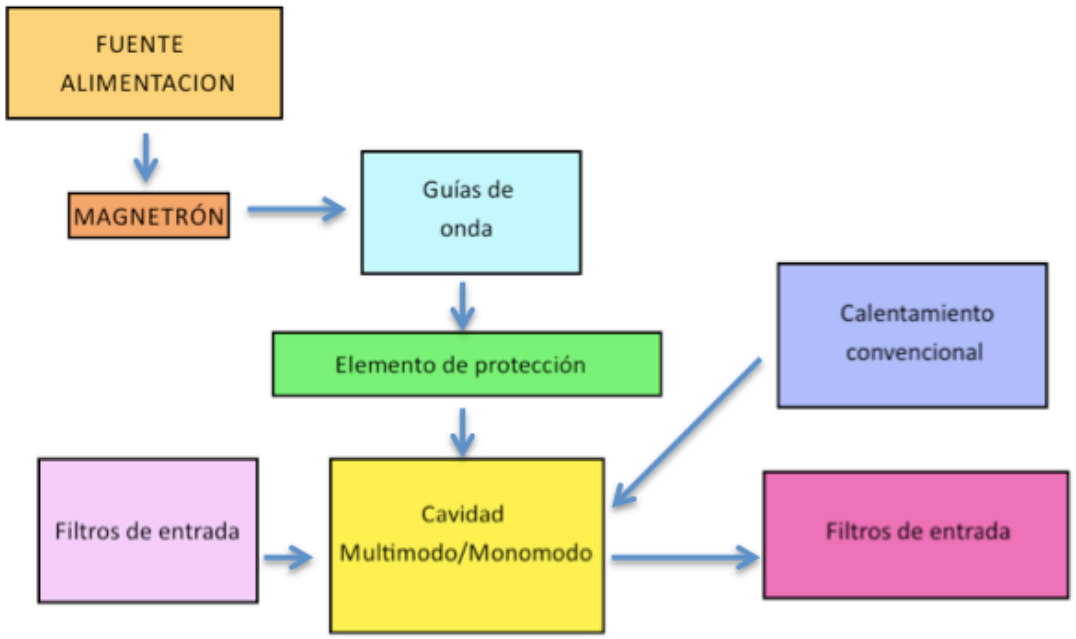

Figura 12: Diagrama de un sistema de calentamiento por microondas básico.

A continuación, se explicarán brevemente cada uno de los bloques.

La fuente de alimentación: Se encarga de proporcionar al magnetrón la energía AC/DC necesaria para su funcionamiento. Es un elemento que permite transformar la corriente de red en voltajes y corrientes adecuados para el magnetrón. Sus corrientes de salida son extremadamente altas, por lo que es muy importante manejarlas con precaución.

Magnetrón: Utiliza la energía que se le aporta y la transforma en energía de microondas debido a su estructura resonante. Sus dimensiones permiten controlar la frecuencia de la señal que emite. El control de la potencia puede realizarse controlando, principalmente, la corriente que llega al ánodo o el campo magnético aplicado a la cavidad que conforma el magnetrón. Al conjunto de la fuente y el

magnetrón se le suele llamar fuente de microondas. Cada fuente suele estar asociada a un puerto acoplado a la cavidad de microondas.

Guías de onda: La energía de microondas del magnetrón se acopla a una guía de onda a través de una antena. Esta transmisión se la denomina 'launcher' (lanzador). En los sistemas de calentamiento por microondas se usan casi siempre guías de onda con tamaños normalizados, ya que las potencias que se usan son de cientos o miles de vatios.

Elemento de protección: Constituidos principalmente por circuladores. Permiten alargar la vida de los magnetrones ya que se pueden producir fuertes reflexiones en el sistema.

Cavidad multimodo/monomodo: Proporciona una distribución de campo eléctrico lo más uniforme posible en las muestras que se procesan. Para ello se pueden usar diferentes técnicas como agitadores de modos, movimiento de la muestra…

Existen muchos tipos de aplicadores o cavidades de microondas, desde las monomodo que suelen ser utilizadas en los laboratorios hasta las multimodo que permiten procesar muestras mucho mayores.

Calentamiento convencional: Se añaden para conseguir perfiles de temperatura uniformes. Como por ejemplo, aire caliente en los hornos de microondas para asegurar que la superficie también se calienta.

Filtros de Entrada/Salida: El filtro se realiza en la propia puerta donde se introducen y se extraen las muestras, la cual no debe tener contacto eléctrico con el horno. Si el horno es de proceso continuo, se deben diseñar otro tipo de filtros como los corrugados, los doblemente corrugados o con 'stubs'.

2.3.3 Magnetrón (Generador de microondas).

El magnetrón es el elemento ideal para la generación de calor, debido a que presenta una gran estabilidad en frecuencia y por su eficiencia que se encuentra entre el 60-70 %.

Las siguientes figuras muestran los componentes internos y externos de un magnetrón. En primer lugar se muestra la sección transversal y posteriormente la sección perpendicular.

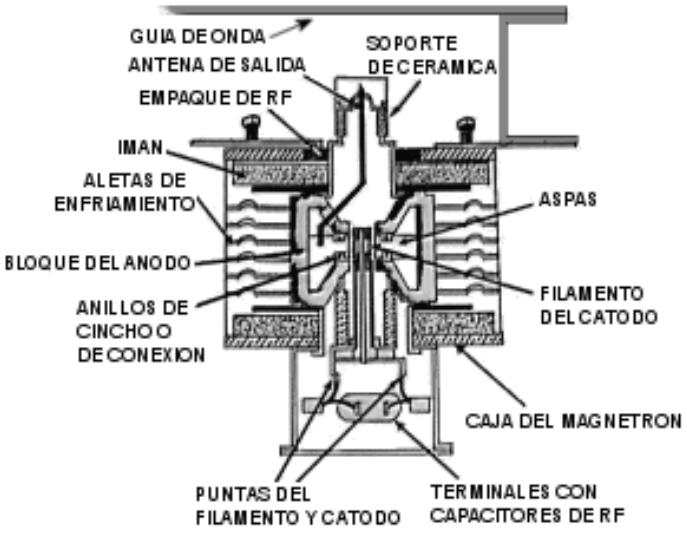

Figura 13: Sección transversal de un magnetrón.

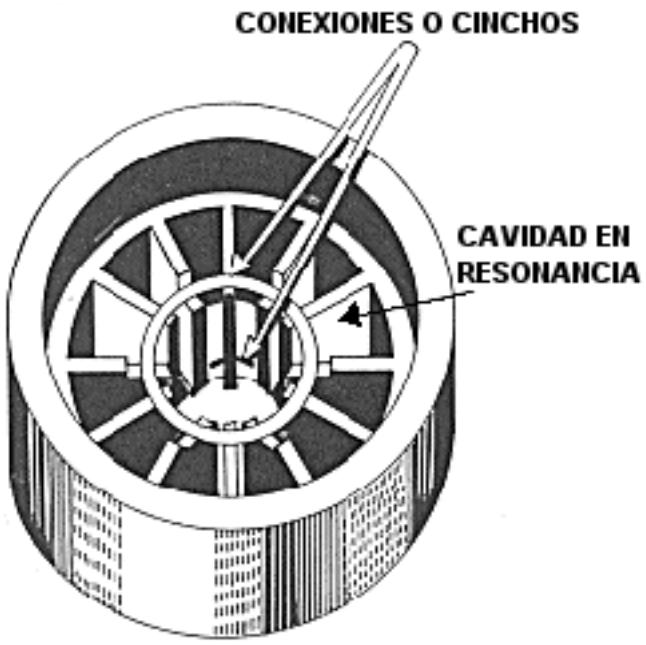

Figura 14: Sección perpendicular de un magnetrón.

El magnetrón está compuesto por una cavidad de microondas en la cual cátodo y ánodo se encuentran a muy diferentes voltajes.

El filamento o cátodo es calentado por las altas corrientes que circulan por él de forma que emite electrones a la cavidad interna. Por otro lado el ánodo tiene una serie de cavidades acopladas entre sí que se proyectan en la dirección radial (también llamadas vanos), de profundidad $\lambda_g/4$ resonando a la frecuencia de operación.

Como el ánodo se encuentra a un voltaje mayor que el cátodo, los electrones emitidos por el cátodo tienen a irse hacia el ánodo, pero el campo magnético generado por el imán impide una trayectoria directa para los electrones y los hace girar en su camino desde el cátodo al ánodo.

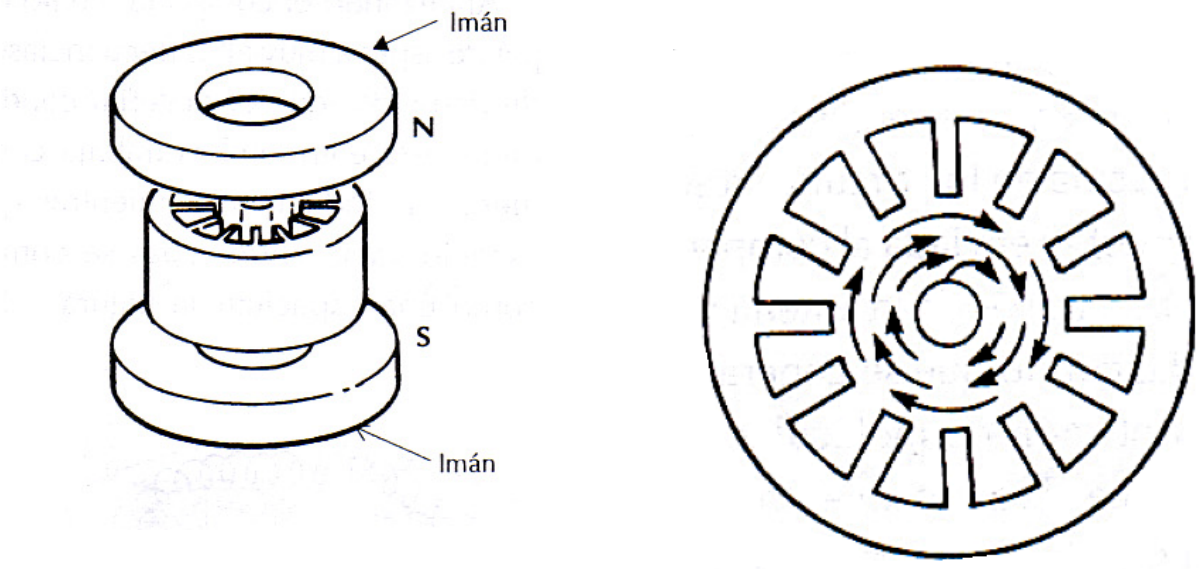

Figura 15: Comportamiento del campo en el Magnetrón.

La energía cinética de los electrones debido a este giro dentro de la cavidad se acopla como energía de microondas en las cavidades debido a que el giro de los electrones no es más que una corriente que genera campos electromagnéticos dentro de la cavidad. El control del voltaje de continua controla la velocidad de los electrones y por tanto la energía de microondas final, al igual que también puede controlarlo la variación del campo magnético en el interior de la cavidad.

Finalmente sólo sería necesario introducir una antena o 'loop' en una de las cavidades para extraer la energía de microondas generada en todas ellas.

Los electrones al impactar con el metal que forma el ánodo liberan energía en forma de calor que no es transformada en energía de microondas. Debido a esto el magnetrón se calienta muy rápido y por consiguiente es necesaria su refrigeración cuando éste se encuentre en funcionamiento. Por debajo de los 2 kilovatios la refrigeración puede ser por aire y para magnetrones con potencias superiores, su refrigeración será por agua.

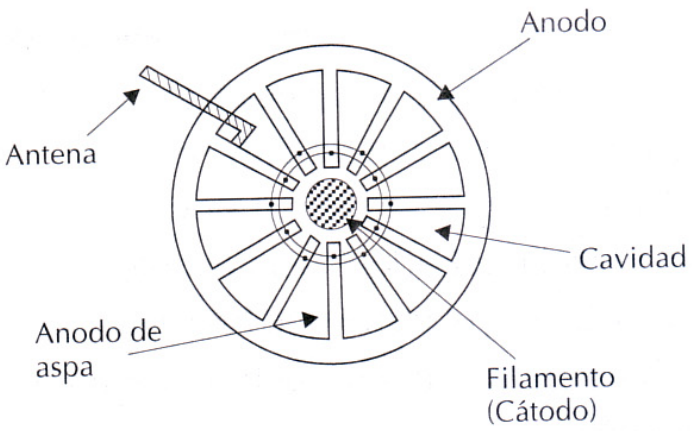

Figura 16: Esquema del Magnetrón con antena incorporada.

Se debe tener en cuenta que su diagrama de operación depende de la relación entre la corriente y el voltaje en el ánodo.

Diagrama de Rieke.

Las variaciones en potencia 'power pulling' y en frecuencia 'frequency pulling' debidas a la carga conectada equivalente a la salida del magnetrón pueden representarse en el Diagrama de Rieke que muestra en la carta de Smith el comportamiento de los magnetrones en función de la impedancia de carga.

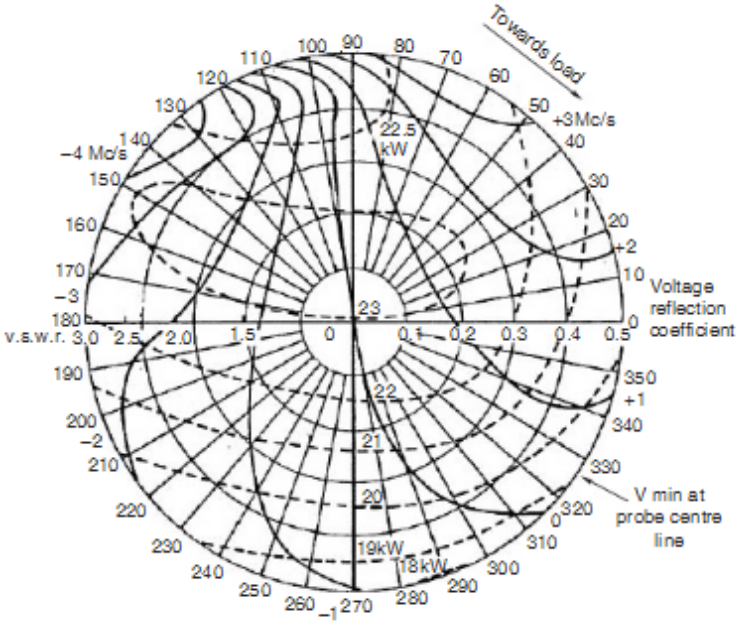

Figura 17: Diagrama de Rieke.

Se muestran la potencia y la frecuencia de salida en función de la carga conectada al magnetrón. Estas curvas de impedancias y reactancias normalizadas intersectan con las curvas de potencia constante y con las curvas que marcan la variación de frecuencia indicando cuáles son las variaciones sufridas por estas magnitudes en función de la impedancia normalizada de la carga.

Dentro de las cargas permitidas por el fabricante, el 'frequency pulling' y el 'power pulling' y suelen estar por debajo de un 0.2% y u 15%. Lo cual supone que para una frecuencia de 2.45 que la frecuencia del magnetrón podrá variar $\pm .9\ MHz$, que mantendría al magnetrón dentro de la banda ICM correspondiente.

Cuando se quiere independizar el funcionamiento del magnetrón de la carga conectada como ya se comentó en el esquema básico del calentamiento por microondas, se suele usar un circulador, que permite trabajar con todas las cargas sin reflexiones. De esta forma, la carga vista del magnetrón es siempre la misma, mejorando así la operación del magnetrón y evitando variaciones indeseadas en potencia y en frecuencia.

2.3.4 Fuentes de alimentación de los magnetrones.

Su función es la de proporcionar una corriente para el ánodo del magnetrón constante independientemente de las fluctuaciones del voltaje de ánodo o de las variaciones de carga, y aplicar el ciclo de trabajo correcto para proporcionar la potencia seleccionada. Para los magnetrones con un campo magnético constante, el voltaje de ánodo debe ser ajustable para dar una corriente de ánodo constante.

Son los dispositivos más caros de un sistema de microondas siendo los responsables de la potencia de salida final del magnetrón. En este apartado se explicarán brevemente 4 tipos de fuentes de alimentación.

1. Fuente con trasformador variable mecánicamente.

En este tipo de fuente, el transformador de entrada es un transformador variable controlado electromecánicamente. Debe existir un servo-control que modifique de forma continua la posición de la aguja en el transformador que modifica la relación de transformación. El servo-control tendrá como señal de entrada una comparación de una tensión de referencia con la corriente de ánodo y permite mantener la potencia de salida del magnetrón seleccionada por el usuario.

En la figura se muestra el esquema básico de la fuente. Una vez fijada la potencia por el usuario, la aguja queda en el transformador variable dejando pasar a la siguiente etapa una porción de la corriente alterna de red. Los diodos rectifican la señal alterna proporcionando una corriente y tensión al magnetrón continua. La corriente de ánodo generará una tensión que se comparará con la tensión de referencia y dependiendo de la comparación la aguja subirá o bajará para ajustarse a la potencia seleccionada por el usuario.

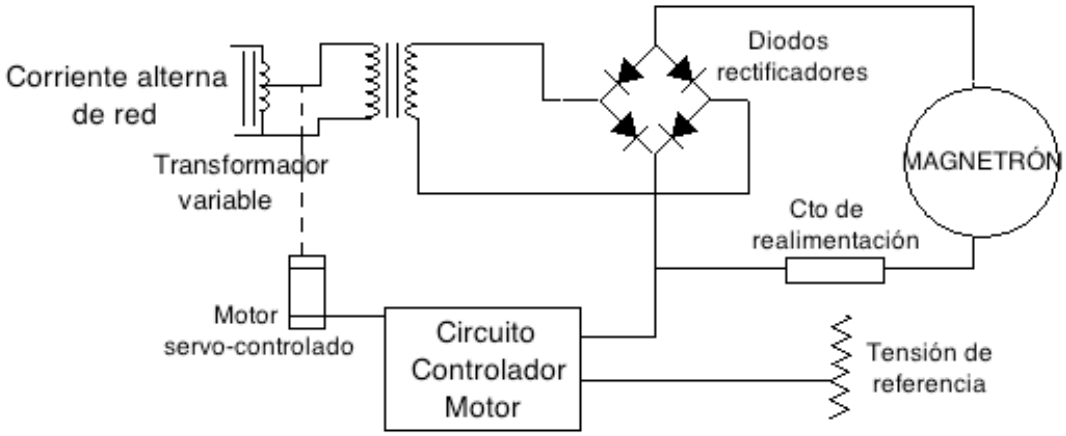

Figura 18: Fuente con transformador variable mecánicamente.

2. Fuente con control por tiristor.

El control de potencia de red que se suministra a los diodos rectificadores se realiza mediante tiristores, siendo fijo en este caso el transformador.

En el sistema se usan tiristores controlados en fase para controlar la potencia de AC al transformador. La diferencia entre el voltaje DC de referencia con el voltaje de la resistencia, se usa para cambiar la fase de los tiristores. Cuando la fase del tiristor es alta, la corriente de salida se activará más tarde por lo que la potencia de salida en el tiristor será menor que la de entrada. Por el contrario cuando la fase de disparo es cero, la potencia de salida en el tiristor será la misma que la de la entrada.

Como el ajuste en este caso es electrónico, se puede realizar más rápido que con el sistema mecánico, pero los tiristores deben de ser capaces de soportar la potencia total de la fuente sin degradarse, o cual obliga a usar dispositivos de muy alta potencia.

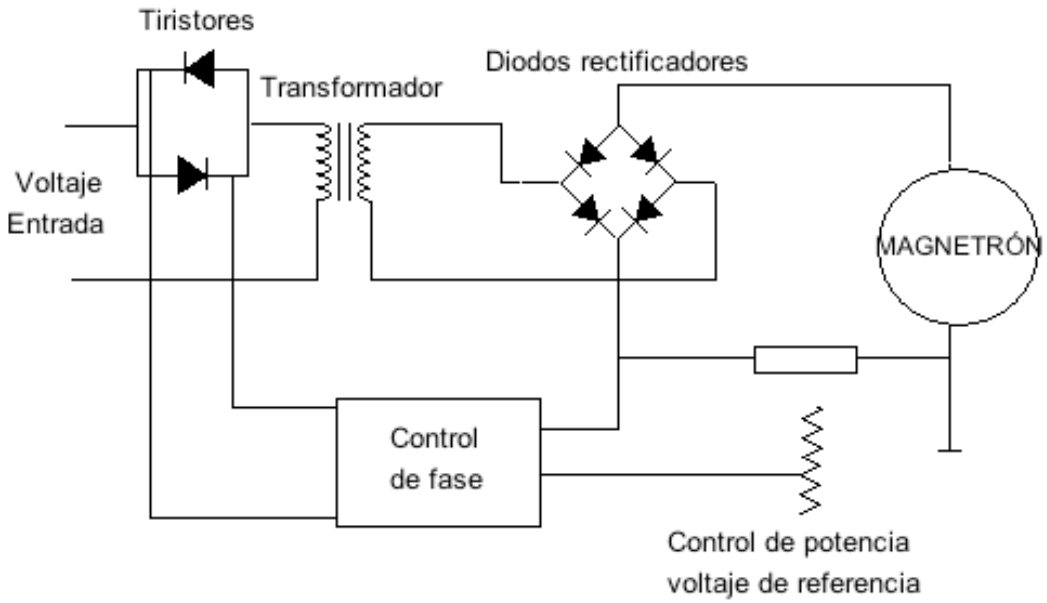

Figura 19: Fuente con control por tiristor.

3. Fuente con control por reactor saturable.

Método muy usado para el control de magnetrones de baja potencia. Usa un inductor variable en serie con un condensador, lo cual provoca un filtro paso banda de primer orden que permite pasar la señal cuando está correctamente sintonizado (cuando se encuentra en resonancia).

En este caso, la inductancia varía con los voltajes crecientes, que hace que se incremente la frecuencia de resonancia del resonador serie y se produzca un decremento de la corriente de ánodo.

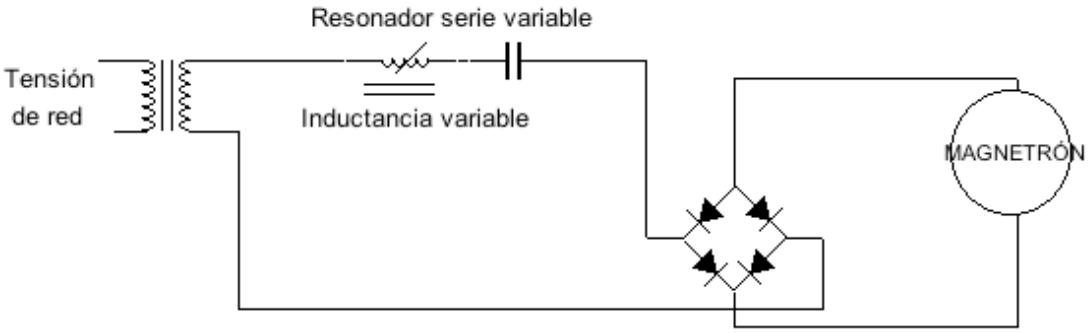

Figura 20: Fuente con control por reactor saturable.

4. Fuente con control por campo magnético variable.

En este tipo de fuentes, la potencia de salida del magnetrón se controla a través del campo magnético aplicado a través del magnetrón.

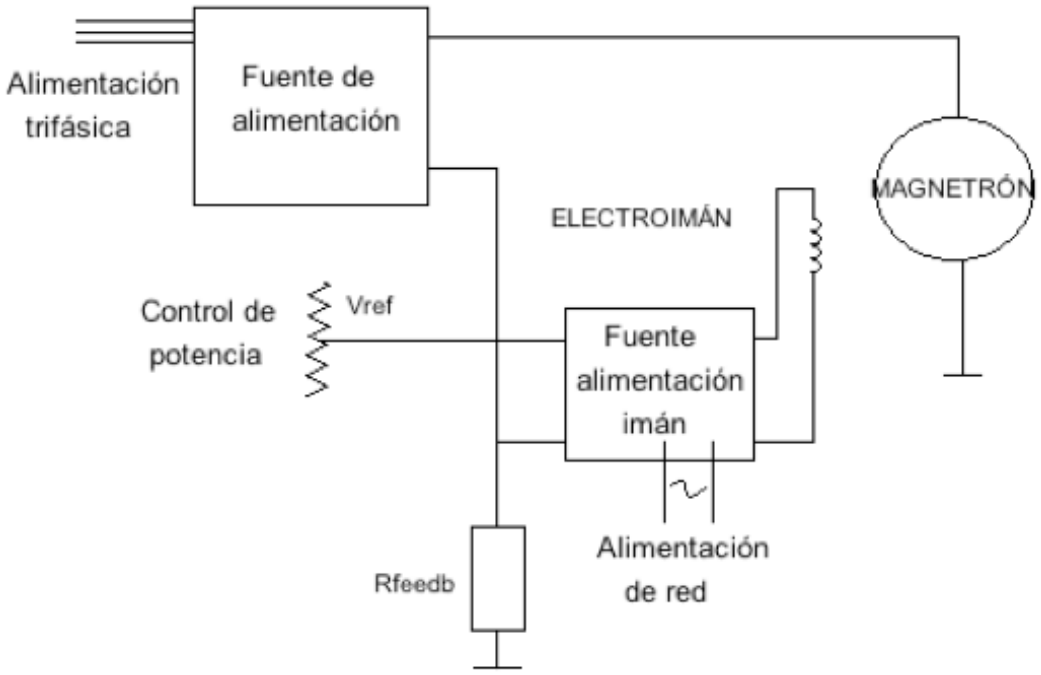

Figura 21: Fuente con control por campo magnético variable.

En el esquema que se muestra en la figura, el control de la intensidad de campo magnético se aplica directamente sobre el magnetrón y el control para mantener dicho campo eléctrico se realiza a través de la señal de referencia y un circuito de control.

La ventaja fundamental de este tipo de fuentes, se da en los magnetrones de alta potencia (20-30 Kw) ya que permite alcanzar grandes potencias sin requerir el control de intensidades de corrientes demasiado elevadas.

3. Descripción de la interfaz de fuente MAGDRIVE1000.

3.1 Introducción.

En este apartado se van a explicar detenidamente cada una de las distintas interfaces programadas en Matlab para el control de la Fuente Magdrive1000.

Todas la interfaces tienen su versión ejecutable tanto para sistemas operativos Mac OS X como para Windows. Esto es debido a que cuando se planteó el proyecto, interesaba que los usuarios a los que va destinado el programa tuviesen la facilidad de poder ejecutarlo sin problemas en ambos entornos.

En este apartado las capturas de pantalla de las interfaces con de la versión MAC OS X. Una vez que se hayan explicado cada una de ellas de forma detallada, se mostrará también el aspecto de las versiones para Windows.

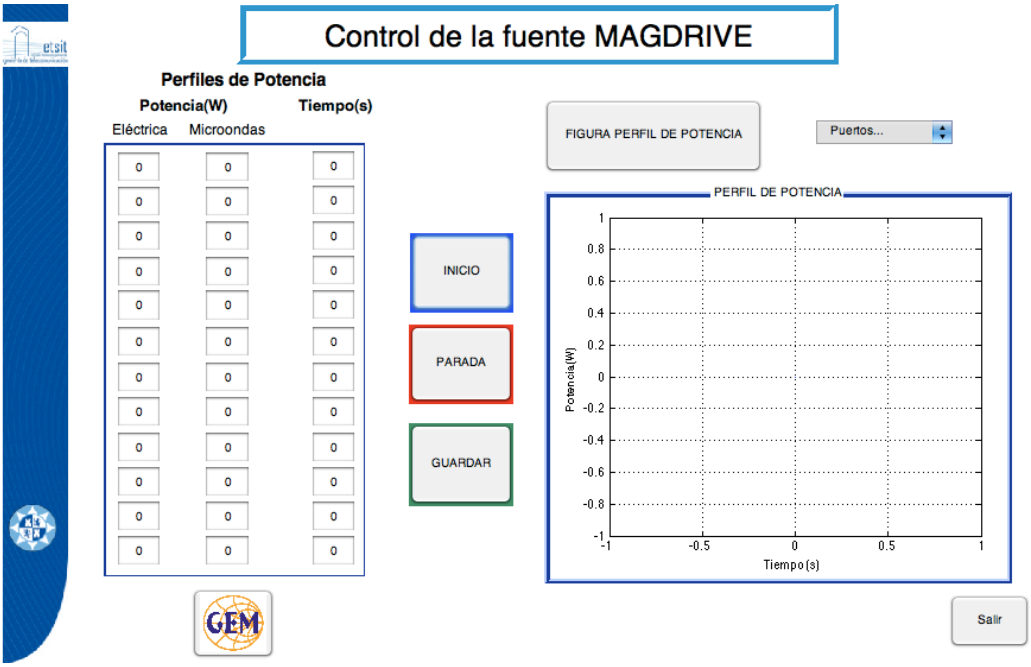

Figura 22: Interfaz Simple.

En primer lugar se explicará la interfaz simple profundizando en ella más en cada uno de sus componentes que en el resto, ya que la mayoría de los elementos introducidos en la interfaz simple se repiten en las otras dos.

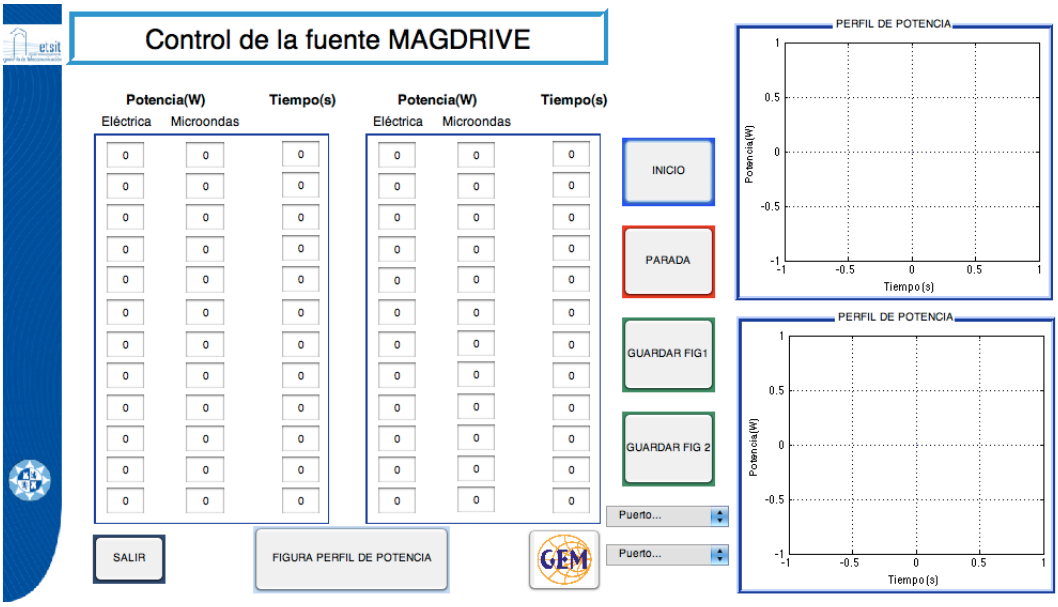

Figura 23: Interfaz Puertos.

En segundo lugar se expondrá la interfaz con dos puertos de comunicación con la que se podrán usar dos fuentes de alimentación.

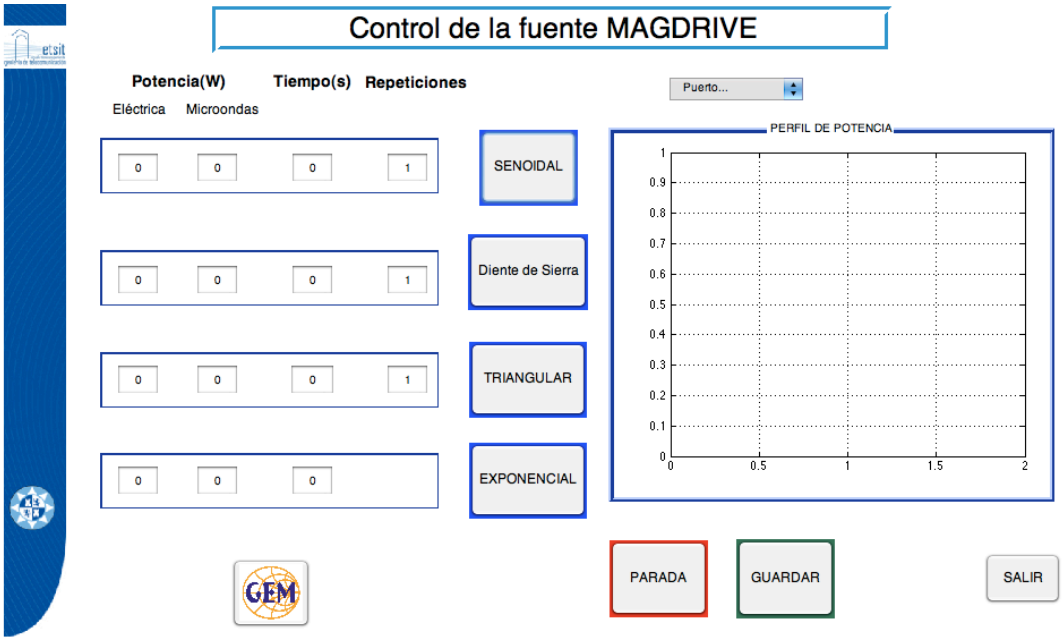

Figura 24: Interfaz Funciones.

Por último se explicará la interfaz que genera cuatro patrones de potencia distintos dependiendo de la función que se seleccione.

3.2 Interfaz Simple.

Es la primera interfaz que se generó. El aspecto de la interfaz simple señalando cada uno de los elementos se muestra en la siguiente figura:

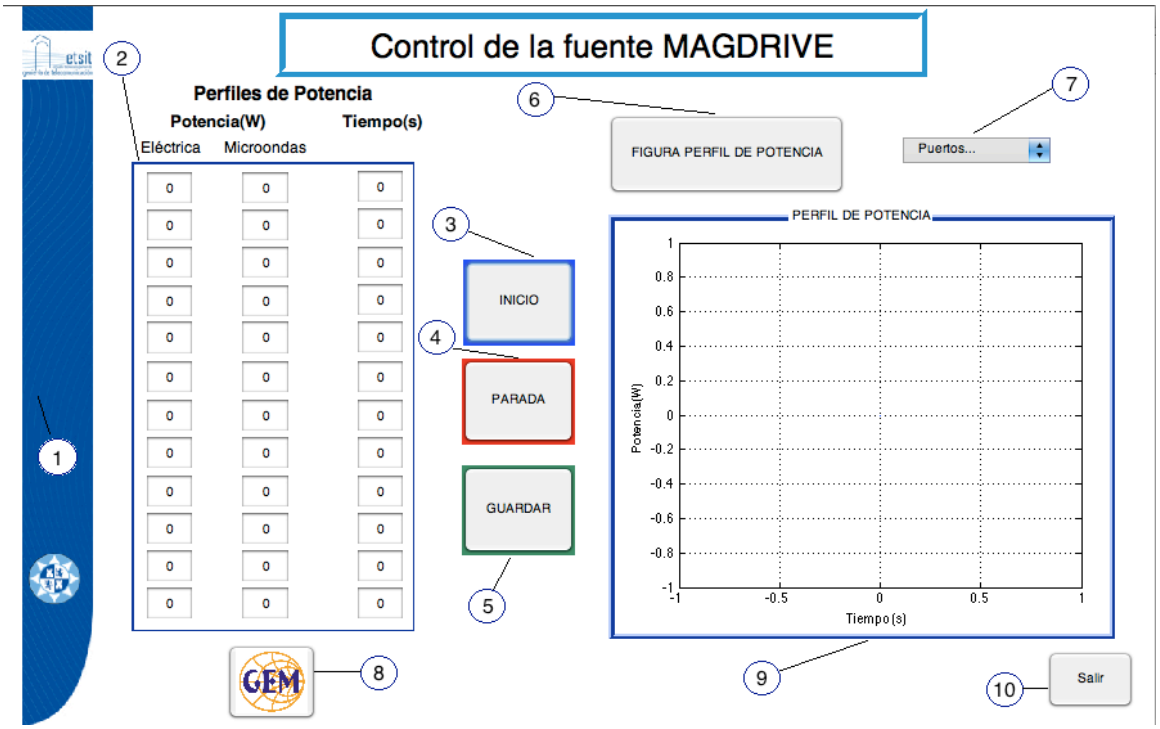

Figura 25: Interfaz Simple. (Elementos)

Los elementos enumerados en la interfaz son los siguientes:

1. Fondo Interfaz UPCT.
2. Tabla Perfiles de Potencias: Potencia eléctrica, Potencia de microondas y Tiempo.
3. Botón de pulsación 'INICIO'.
4. Botón de pulsación 'PARADA.
5. Botón de pulsación 'GUARDAR.
6. Botón de pulsación 'FIGURA PERFIL DE POTENCIA'.
7. Lista desplegable para la selección de puertos de comunicación con la fuente.
8. Botón de pulsación de carácter informativo 'GEM'.
9. Figura que representa el perfil de potencia a transmitir.
10. Botón de pulsación 'Salir'.

A continuación se expondrán de forma detallada de cada una de las partes más importantes de la interfaz.

En primer lugar se puede observar una tabla denominada 'Perfiles de Potencia' , que está formada por tres columnas marcada con el número 2:

- La primera columna representa la Potencia eléctrica introducida. Está formada por 12 cajas de texto editables, los cuales se encuentran inicializados todos a 0 vatios.

- La segunda columna representa la Potencia en microondas. Está también constituida por 12 cajas de texto, pero en este caso no son cajas de texto editables por el usuario, ya que dependen de los valores de potencia eléctrica introducidos en la primera columna. Siguen la siguiente relación 1300 W potencia eléctrica → 1000 W potencia de microondas.

- La tercera columna representa los distintos instantes de tiempo durante los que se transmitirá el valor de potencia deseado. En este caso son 12 cajas de texto editables por el usuario, inicializados todos a 0 segundos.

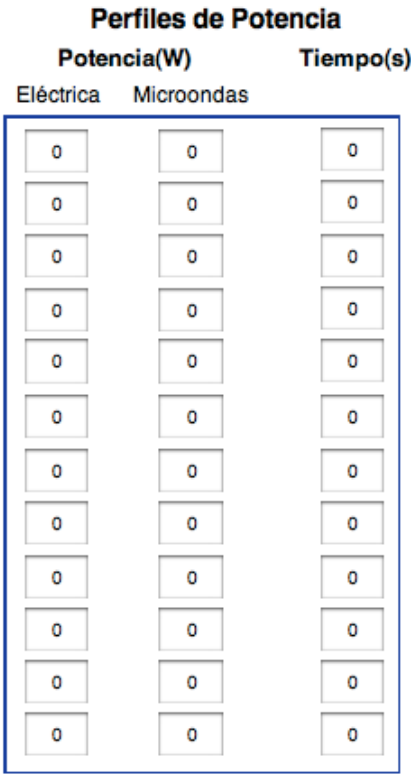

Figura 26: Perfiles de Potencia.

A la derecha de la tabla, se encuentran tres botones de selección denominados: Inicio (número 3), Parada (número 4) y Guardar (número 5).

Figura 27: Botón INICIO.

Inicio:

Su función tal y como su nombre indica es la de iniciar la comunicación con el fuente MAGDRIVE1000. Procesa los datos de potencia y tiempos introducidos en la tabla de perfiles de potencia, generando dos vectores.

Los valores que se introducen deben ser:

- Valores numéricos positivos.
- $\geq$ 80 (W).
- $\leq$ 1300 (W).

Si alguno de los valores introducidos por el usuario no cumple las condiciones anteriores, se abre una ventana que comunica el error al usuario y cambia el valor automáticamente a uno válido. A continuación se muestran en las siguientes figuras los tres tipos de errores que el usuario puede cometer.

Figura 28: Ventanas de ERROR.

- El primer error aparece si se introduce un valor de potencia $\leq$ 80 (W).
- El segundo error aparece si se introduce un valor de potencia es $\geq$ de 1300 (W).
- El tercer error aparece si se introduce un valor negativo o un valor no numérico.

El primero de los vectores, será un vector de tiempos de tamaño variable dependiendo de los tiempos introducidos. El segundo de los vectores será un vector de potencias cuyo tamaño será el mismo que el vector de tiempos en el que el valor de las potencias a transmitir cambiará dependiendo del instante de tiempo en que nos encontremos.

Una vez creados estos vectores se abrirán los puertos de comunicación de la fuente Magdrive1000 activando la transmisión de potencia de la fuente. Cuando se terminen de transmitir todas las órdenes a la fuente, como medida de seguridad se enviará un comando de apagado a la fuente para evitar que ésta siga funcionando.

Se debe tener en cuenta que en todo momento además de comprobar la condición de parada que el usuario puede introducir en cualquier momento al pulsar el botón de 'Parada' (el cual se explica a continuación), también se comprueban los estados de alarma de la fuente, que pueden ser los siguientes:

- Fallo, control del Filamento'.
- Fallo, control de Potencia Anódica.
- Fallo, control de Temperatura.
- N.C.
- Regulación de Temperatura en progreso.
- Fallo, control PFC.
- Unidad deshabilitada debido a una alarma.

Al pulsar el Botón de Inicio, además de iniciarse la transmisión, se mostrará automáticamente en el recuadro 'axes' denominado 'PERFIL DE POTENCIA', una representación gráfica del perfil de potencia que se va a transmitir.

Además de enviar datos a la fuente para que ésta transmita las potencias deseadas, el ordenador también lee de la fuente y comprueba que cada dato que se envía es recibido dejando un tiempo de 0.94 segundos en cada transmisión y comprobación.

Figura 29: Botón PARADA.

Parada:

Su función es la de poder detener la transmisión de potencia en cualquier momento.

Cuando se pulsa el botón, se abre automáticamente un cuadro de diálogo en el que se le pregunta al usuario si desea detener la transmisión. Si el usuario selecciona 'NO', se cierra el cuadro de diálogo continuando la transmisión. Si el usuario selecciona 'SI', se cierra el cuadro de diálogo y se detiene la transmisión.

Esto se debe a que por defecto se ha establecido dentro del código que la condición de

parada es 'NO', cuyo estado se comprueba en cada iteración del bucle 'for' antes de transmitir la potencia correspondiente en cada instante.
El código correspondiente a esta comprobación, se puede encontrar en el anexo dentro de la función 'function Inicio_Callback'.

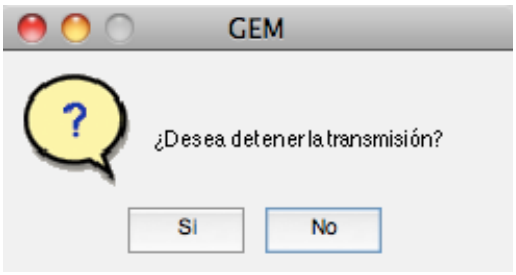

Figura 30: Condición Parada

Figura 31: Botón GUARDAR

Guardar:

La función del botón Guardar es la de poder salvar la figura del perfil de potencia que se ha generado. Para ello se abre automáticamente una nueva ventana con el perfil de potencia introducido, en la que se encuentra la opción de salvar la figura en diferentes formatos tal y como se pueden ver en las siguientes capturas.

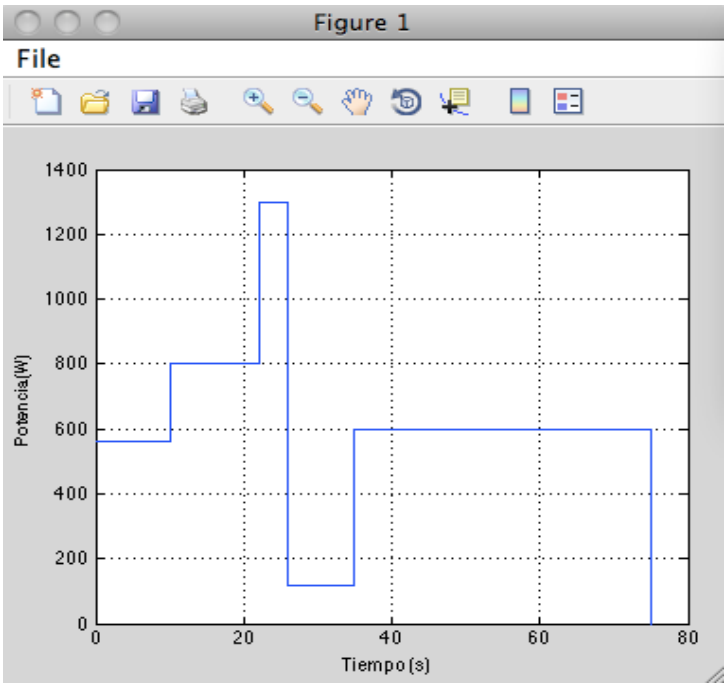

Figura 32: Perfil de potencia aleatorio

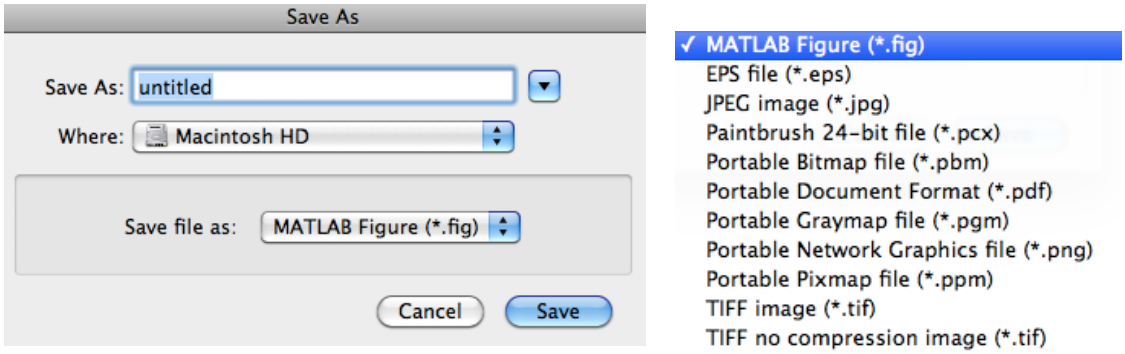

Figura 33: Cuadro diálogo guardar figura

Además de los botones de selección de Inicio, Parada y Guardar, en la interfaz se encuentran otros botones: 'FIGURA PERFIL DE POTENCIA' (número 6), 'GEM' (número 8) y 'Salir'(número 10).

Figura 34: Botón FIGURA PERFIL DE POTENCIA

FIGURA PERFIL DE POTENCIA:

El botón 'Figura perfil de potencia', se usa para obtener una previsualización de la figura del perfil de potencia que se va a transmitir, la cual se mostrará en el recuadro 'axes'.

De esta forma el usuario puede comprobar antes de iniciar la transmisión el perfil de potencia que desea transmitir, evitando así tener que usar el botón de 'Parada' debido a un error al introducir algún valor indeseado.

El código usado para generar la figura del perfil de potencia es el mismo que usa al pulsar el botón 'Inicio' omitiendo la parte de código que se usa para la transmisión.

Figura 35: Botón GEM

GEM:

Este botón sólo es de carácter informativo y al pulsarlo se abre una ventana que muestra el grupo de investigación en el cual se ha desarrollado la interfaz.

Figura 36: Diálogo botón GEM

Figura 37: Botón Salir

Salir:

El siguiente botón se usa para salir de interfaz una vez finalizado su uso. Al pulsarlo se muestra el siguiente cuadro de diálogo dónde el usuario decide si desea seguir usando el programa o desea salir de él.

Figura 38: Diálogo botón Salir

Por último en la interfaz, se ha introducido un 'Popup Menu' desplegable (número 7) en el que se puede seleccionar el puerto de comunicación que se va a usar entre el ordenador y la fuente MAGDRIVE1000.

Figura 39: Selección puerto de comunicación.

3.3 Interfaz Puertos.

La segunda interfaz programada para el control de la fuente MAGDRIVE1000 es la que se muestra en la siguiente figura con sus elementos señalados.

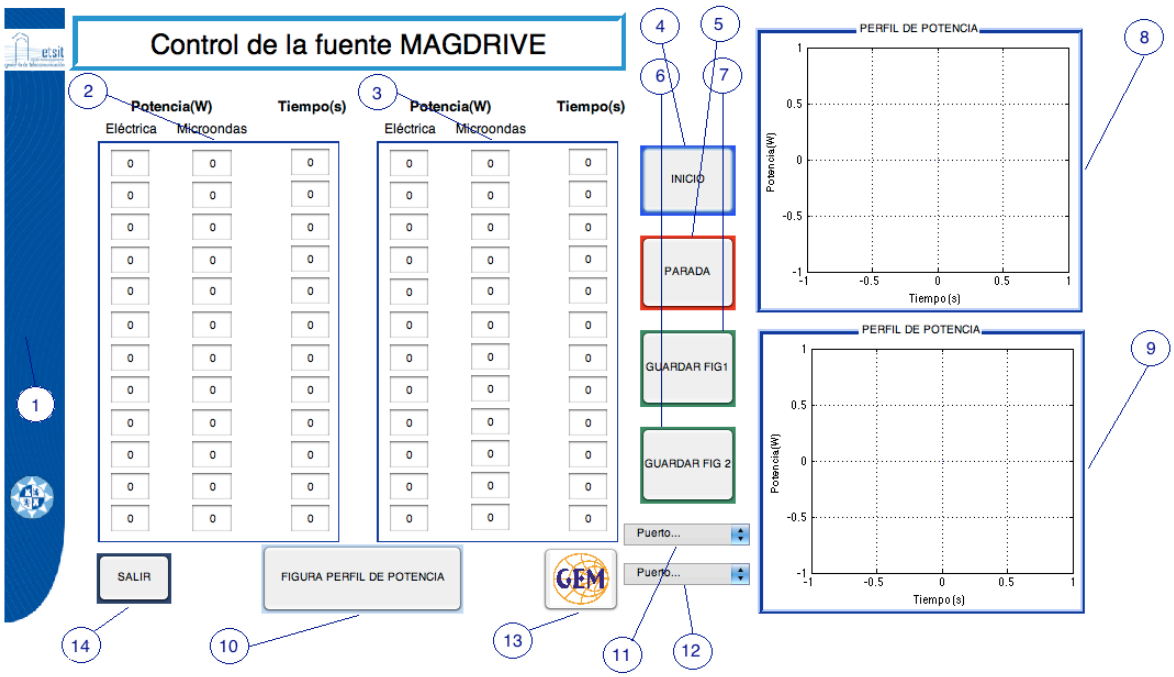

Figura 40: Interfaz puertos. (Elementos)

Los elementos enumerados en la interfaz son los siguientes:

1. Fondo Interfaz UPCT.
2. Tabla Perfiles de Potencias 1: Potencia eléctrica, Potencia de microondas y Tiempo.
3. Tabla Perfiles de Potencias 2: Potencia eléctrica, Potencia de microondas y Tiempo.
4. Botón de pulsación 'INICIO'.
5. Botón de pulsación 'PARADA.
6. Botón de pulsación 'GUARDAR FIG1'.
7. Botón de pulsación 'GUARDAR FIG2'.
8. Figura que representa el perfil de potencia 1 a transmitir.
9. Figura que representa el perfil de potencia 2 a transmitir.
10. Botón de pulsación 'FIGURA PERFIL DE POTENCIA'.
11. Lista desplegable para la selección de puertos de comunicación con Fuente1.
12. Lista desplegable para la selección de puertos de comunicación con Fuente2.
13. Botón de pulsación de carácter informativo 'GEM'.
14. Botón de pulsación 'Salir.

Si se observa la interfaz, se puede apreciar que se sigue el mismo modelo de la interfaz simple, sin embargo en ésta, casi todos los elementos se encuentran por duplicado para poder transmitir diferentes patrones de potencia y poder controlar a la vez dos fuentes.

En este modelo, los vectores de potencia y tiempo correspondientes a cada uno de los perfiles de potencia se van transmitiendo alternativamente de tal manera que las dos fuentes que se desean usar se encuentran en funcionamiento a la vez, pudiendo así realizar ambas transmisiones de forma simultánea.

Una vez introducidos las potencias y los tiempos deseados para ambos perfiles de potencia, se deben seleccionar los puertos de comunicación entre ordenador y las fuentes.

Figura 41: Selección de dos puertos de comunicación.

Cuando se han seleccionado los puertos de comunicación, se pulsa el botón 'INICIAR' para comenzar el uso de ambas fuentes. A continuación se explicará detenidamente el proceso mediante el cual se transmiten los valores a las fuentes.

1. En primer lugar, se procesan los datos introducidos de la misma forma que en la interfaz simple, pero generando en este caso como es lógico, dos vectores de tiempo y dos vectores de potencia.

2. Una vez creados los vectores se muestran en 'axes1' y 'axes2' llamados 'Perfil de potencia', las gráficas correspondientes a cada uno de los perfiles.

3. En la interfaz simple, sólo era necesario el vector de potencias para la transmisión. Sin embargo en este caso, como se quieren transmitir dos perfiles de potencia distintos, es necesario generar otros dos nuevos vectores denominados 'fuente_1' y 'fuente_2', los cuales podrán ser del mismo, o de diferente tamaño dependiendo de los tiempos introducidos por el usuario para cada una de las potencias a transmitir.

4. Una vez creados los vectores fuente, se comparan y se iguala el tamaño de ambos introduciendo ceros en el que sea de menor tamaño. Este paso es fundamental para poder enviarlos de forma simultánea evitando de esta manera errores debido a una diferencia en la longitud de los vectores.

5. A continuación, se abren los puertos de comunicación elegidos para cada una de las transmisiones entre el ordenador y las fuentes. Dentro del bucle *for* como puede observarse en el código del anexo, se recorrerán por completo los

vectores fuente, enviando en cada iteración los valores a la fuente MAGDRIVE1000.

6. Finalmente una vez realizada la transmisión, se envían los comandos de apagado para cada una de las fuentes y se cierra la comunicación de los puertos serie.

3.4 Interfaz Funciones.

Es la última de las tres interfaces programadas para el control de la fuente MAGDRIVE1000. En la siguiente figura se muestra la interfaz con sus elementos más importantes señalados.

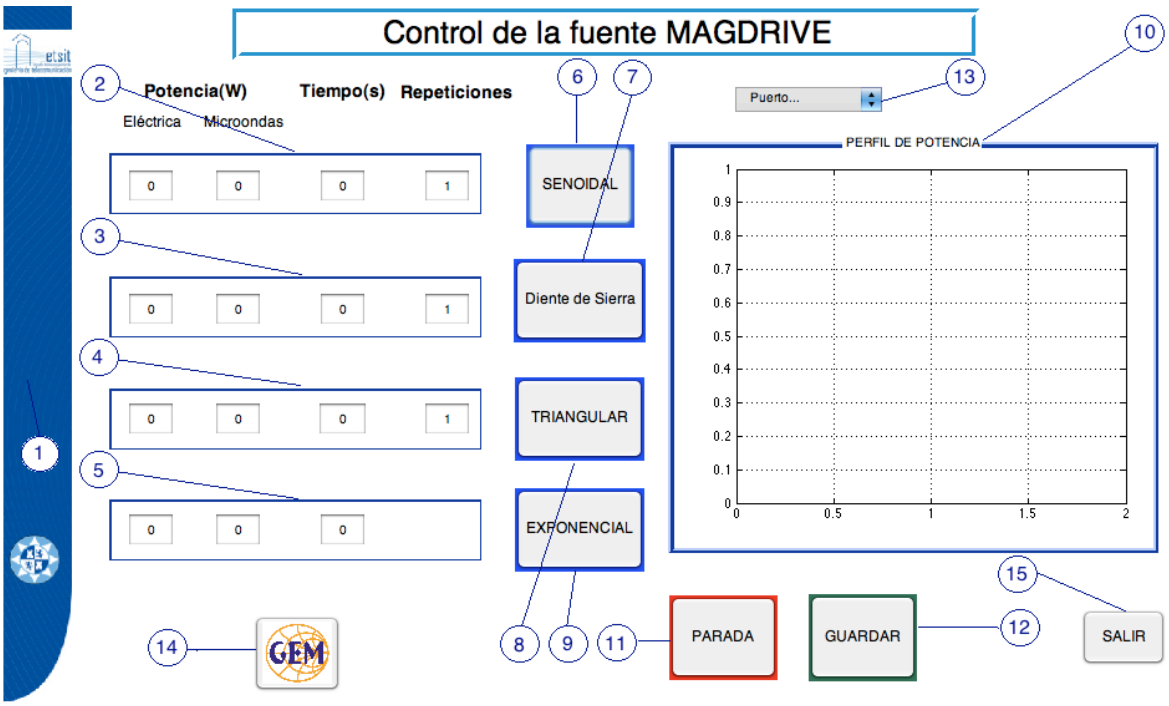

Figura 42: Interfaz Funciones. (Elementos)

Los elementos enumerados en la interfaz son los siguientes:

1. Fondo Interfaz UPCT.
2. Tabla Perfiles de Potencias Distribución Senoidal: Potencia eléctrica, Potencia en microondas, Tiempo y Repeticiones.
3. Tabla Perfiles de Potencias Distribución Diente de Sierra: Potencia eléctrica, Potencia en microondas, Tiempo y Repeticiones.
4. Tabla Perfiles de Potencias Distribución Triangular: Potencia eléctrica, Potencia en microondas, Tiempo y Repeticiones.
5. Tabla Perfiles de Potencias Distribución Senoidal: Potencia eléctrica, Potencia en microondas y Tiempo.
6. Botón de pulsación 'SENOIDAL'.
7. Botón de pulsación 'DIENTE DE SIERRA'.
8. Botón de pulsación 'TRIANGULAR'.
9. Botón de pulsación 'EXPONENCIAL'.
10. Figura que representa el perfil de potencia a transmitir.
11. Botón de pulsación 'PARADA'.

12. Botón de pulsación ‘GUARDAR’.
13. Lista desplegable para la selección de puertos de comunicación con la fuente.
14. Botón de pulsación de carácter informativo ‘GEM’.
15. Botón de pulsación ‘Salir.

En la Interfaz Funciones se pueden general perfiles de potencia siguiendo cuatro distribuciones: Senoidal, Diente de Sierra, Triangular y Exponencial.

A cada una de las distribuciones le corresponden un Botón de pulsación el cual tendrá la misma función que el Botón de pulsación ‘INICIO’ ya comentado en las interfaces anteriores y una tabla en la que introducir la potencia eléctrica, el tiempo y el número de repeticiones. A continuación se explicarán de una forma más concreta cada uno de los nuevos perfiles de potencia que se pueden generar con esta interfaz.

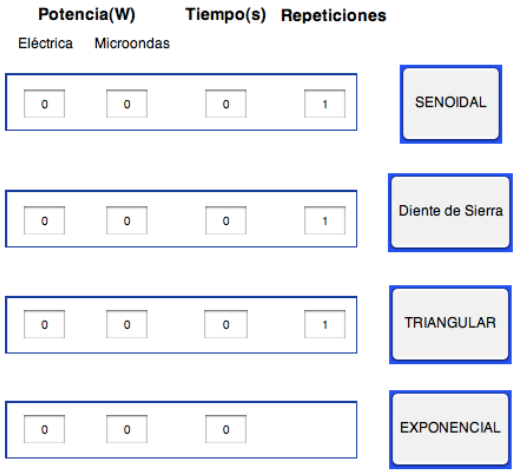

Figura 43: Interfaz Funciones. (Perfiles de Potencia)

1. Distribución Senoidal.

Si se desea crear un patrón de potencias que siga una distribución senoidal, el usuario deberá introducir en la Tabla Perfiles de Potencia (número 2) un valor máximo de potencia eléctrica, el tiempo que se desea que dure el periodo de cada pulso y el número de repeticiones del pulso.

Debido a que a la fuente no se pueden transmitir valores negativos, la función de distribución se ha programado en valor absoluto, para que siempre se obtengan valores positivos. La condición de valor absoluto se ha seguido también en el resto de distribuciones por lo que todos los resultados en el resto de apartados también serán positivos.

Por ejemplo si se introducen los siguiente valores:

- $Potencia\ eléctrica\ =\ 500\ (W)$.
- $Tiempo\ =\ 10\ (s)$.

- $Repeticiones = 2$.

Figura 44: Ejemplo interfaz Senoidal. (Datos)

El resultado obtenido será el siguiente:

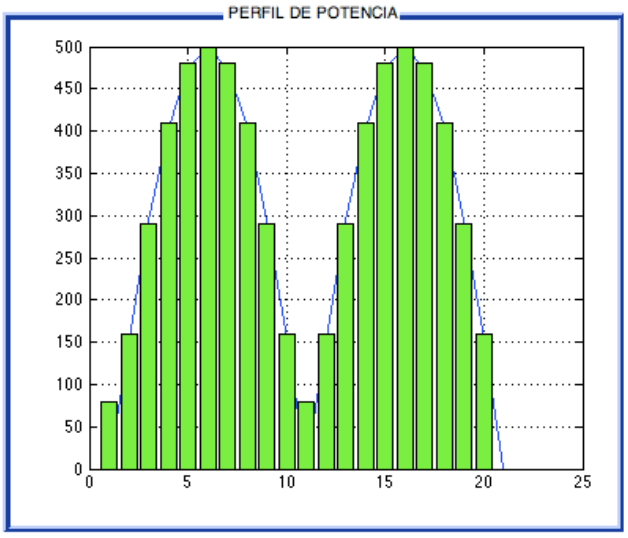

Figura 45: Ejemplo interfaz Senoidal. (Perfil de Potencia)

Como se puede observar el perfil de potencia que la fuente va a transmitir será el de dos pulsos senoidales en un tiempo total de 20 segundos.

El funcionamiento interno una vez que se pulsa el botón 'SENOIDAL' es el siguiente:

1. En primer lugar, se procesan los datos introducidos de la misma forma que en la interfaz simple.

2. Se creará un eje de tiempos en función del tiempo introducido y el número de repeticiones deseadas por el usuario.

3. Como puede verse referenciado en el anexo `function seno_Callback`, se creará un vector llamado 'x' que estará formado por los valores correspondientes con el valor absoluto del un pulso senoidal y que posteriormente formará el vector 'a' cuyos valores serán los mismos que los de 'x' con las restricciones de potencia máxima y mínima de la fuente MAGDRIVE1000. Los datos serán repetidos tantas veces como indique el número de repeticiones, siendo éstos los valores que se transmitirán a la fuente.

4. Como en las otras interfaces, se muestra el Perfil de potencia en pantalla.

5. Se abren los puertos de comunicación elegidos para la transmisión entre el ordenador y la fuente, y se envía en cada iteración del bucle *for* los valores a la fuente MAGDRIVE1000.

6. Finalmente una vez realizada la transmisión, se envían los comandos de apagado a la fuente y se cierra la comunicación del puerto serie.

2. Distribución Diente de Sierra.

Si se desea crear un patrón de potencias que siga una distribución Diente de Sierra, el usuario deberá introducir en la Tabla Perfiles de Potencia (número 3) un valor máximo de potencia eléctrica, el tiempo que se desea que dure el periodo de cada pulso y el número de repeticiones del pulso.

Por ejemplo si se introducen los siguiente valores:

- $Potencia\ eléctrica\ =\ 700\ (W)$.
- $Tiempo\ =\ 15\ (s)$.
- $Repeticiones\ =\ 3$.

Figura 46: Ejemplo interfaz Diente de Sierra. (Datos)

El resultado obtenido será el siguiente:

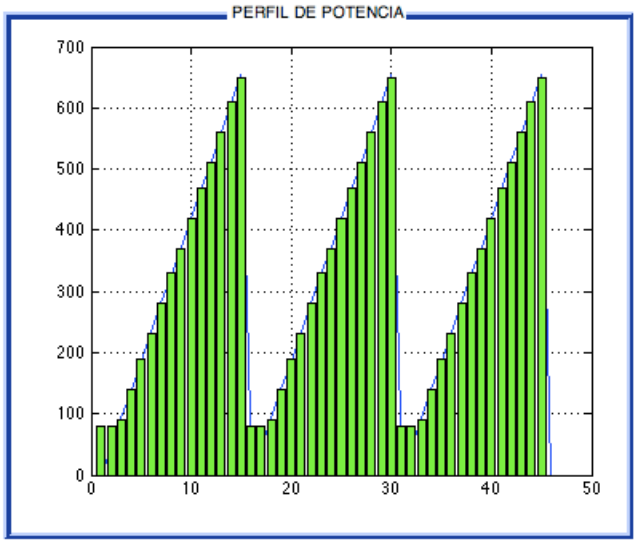

Figura 47: Ejemplo interfaz Diente de Sierra. (Perfil de Potencia)

Como se puede observar el perfil de potencia que la fuente va a transmitir será el de tres pulsos diente de sierra en un tiempo total de 45 segundos.

El funcionamiento interno una vez que se pulsa el botón 'Diente de Sierra' es el siguiente:

1. En primer lugar, se procesan los datos introducidos de la misma forma que en la interfaz simple.

2. Se creará un eje de tiempos en función del tiempo introducido y el número de repeticiones deseadas por el usuario.

3. Como puede verse referenciado en el anexo `function sierra_Callback` , se creará un vector llamado 'x' que estará formado por los valores correspondientes al pulso diente de sierra y que posteriormente formará el vector 'a' cuyos valores serán los mismos que los de 'x' con las restricciones de potencia máxima y mínima de la fuente MAGDRIVE1000. Los datos serán repetidos tantas veces como indique el número de repeticiones, siendo éstos los valores que se transmitirán a la fuente.

4. Como en las otras interfaces, se muestra el Perfil de potencia en pantalla.

5. Se abren los puertos de comunicación elegidos para la transmisión entre el ordenador y la fuente, y se envía en cada iteración del bucle *for* los valores a la fuente MAGDRIVE1000.

6. Finalmente una vez realizada la transmisión, se envían los comandos de apagado a la fuente y se cierra la comunicación del puerto serie.

3. Distribución Triangular.

Si se desea crear un patrón de potencias que siga una distribución Triangular, el usuario deberá introducir en la Tabla Perfiles de Potencia (número 4) un valor máximo de potencia eléctrica, el tiempo que se desea que dure el periodo de cada pulso y el número de repeticiones del pulso.

Por ejemplo si se introducen los siguientes valores:

- $Potencia\ eléctrica\ =\ 900\ (W)$.
- $Tiempo\ =\ 20\ (s)$.
- $Repeticiones\ =\ 2$.

Figura 48: Ejemplo interfaz Triangular. (Datos)

El resultado obtenido será el siguiente:

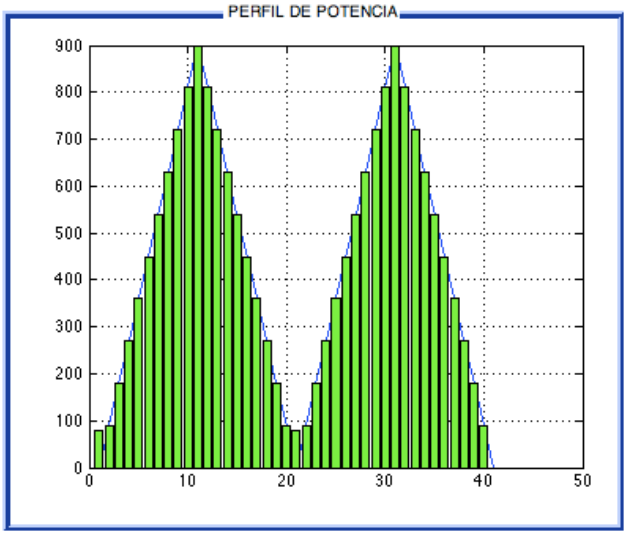

Figura 49: Ejemplo interfaz Triangular. (Perfil de Potencia)

Como se puede observar el perfil de potencia que la fuente va a transmitir, será el de dos pulsos triangulares en un tiempo total de 40 segundos.

El funcionamiento interno una vez que se pulsa el botón 'TRIANGULAR' es el siguiente:

1. En primer lugar, se procesan los datos introducidos de la misma forma que en la interfaz simple.

2. Se creará un eje de tiempos en función del tiempo introducido y el número de repeticiones deseadas por el usuario.

3. Como puede verse referenciado en el anexo `function triangular_Callback`, se creará un vector llamado 'x' que estará formado por los valores correspondientes al pulso triangular y que posteriormente formará el vector 'a' cuyos valores serán los mismos que los de 'x' con las restricciones de potencia máxima y mínima de la fuente MAGDRIVE1000. Los datos serán repetidos tantas veces como indique el número de repeticiones, siendo éstos los valores que se transmitirán a la fuente.

4. Como en las otras interfaces, se muestra el Perfil de potencia en pantalla.

5. Se abren los puertos de comunicación elegidos para la transmisión entre el ordenador y la fuente, y se envía en cada iteración del bucle *for* los valores a la fuente MAGDRIVE1000.

6. Finalmente una vez realizada la transmisión, se envían los comandos de apagado a la fuente y se cierra la comunicación del puerto serie.

4. Distribución Exponencial.

Si se desea crear un patrón de potencias que siga una distribución Exponencial, el usuario deberá introducir en la Tabla Perfiles de Potencia (número 5) un valor máximo de potencia eléctrica y el tiempo. En este tipo de distribuciones realizar repeticiones no tiene sentido ya que al ser valores exponenciales tal y como se verá en el apartado de análisis y resultados los resultados obtenidos no serían buenos debido a que no daría tiempo a que se refrigerase el magnetrón.

Por ejemplo si se introducen los siguiente valores:

- $Potencia\ eléctrica\ =\ 6\ (W)$.
- $Tiempo\ =\ 5\ (s).\ Repeticiones\ =\ 3$.

Figura 50: Ejemplo interfaz Exponencial. (Datos)

El resultado obtenido será el siguiente:

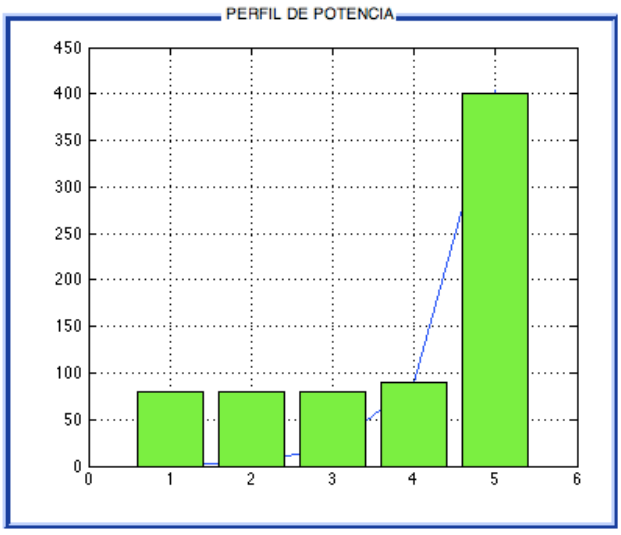

Figura 51: Ejemplo interfaz Exponencial. (Perfil de Potencia)

Como se puede observar el perfil de potencia que la fuente va a seguir una distribución exponencial durante 5 segundos.

El funcionamiento interno una vez que se pulsa el botón 'EXPONENCIAL' es el siguiente:

1. En primer lugar, se procesan los datos introducidos de la misma forma que en la interfaz simple.

2. Se creará un eje de tiempos en dependiendo del tiempo introducido por el usuario.

3. Como puede verse referenciado en el anexo `function expocencial_Callback` , se creará un vector llamado 'x' que estará formado por los valores correspondientes al pulso exponencial y que posteriormente formará el vector 'a' cuyos valores serán los mismos que los de 'x' con las restricciones de potencia máxima y mínima de la fuente MAGDRIVE1000.

4. Como en las otras interfaces, se muestra el Perfil de potencia en pantalla.

5. Se abren los puertos de comunicación elegidos para la transmisión entre el ordenador y la fuente, y se envía en cada iteración del bucle *for* los valores a la fuente MAGDRIVE1000.

6. Finalmente una vez realizada la transmisión, se envían los comandos de apagado a la fuente y se cierra la comunicación del puerto serie.

3.5 Interfaces Windows.

Por último una vez expuestas todas la interfaces, en este apartado se van a mostrar las versiones para el sistema operativo Windows.

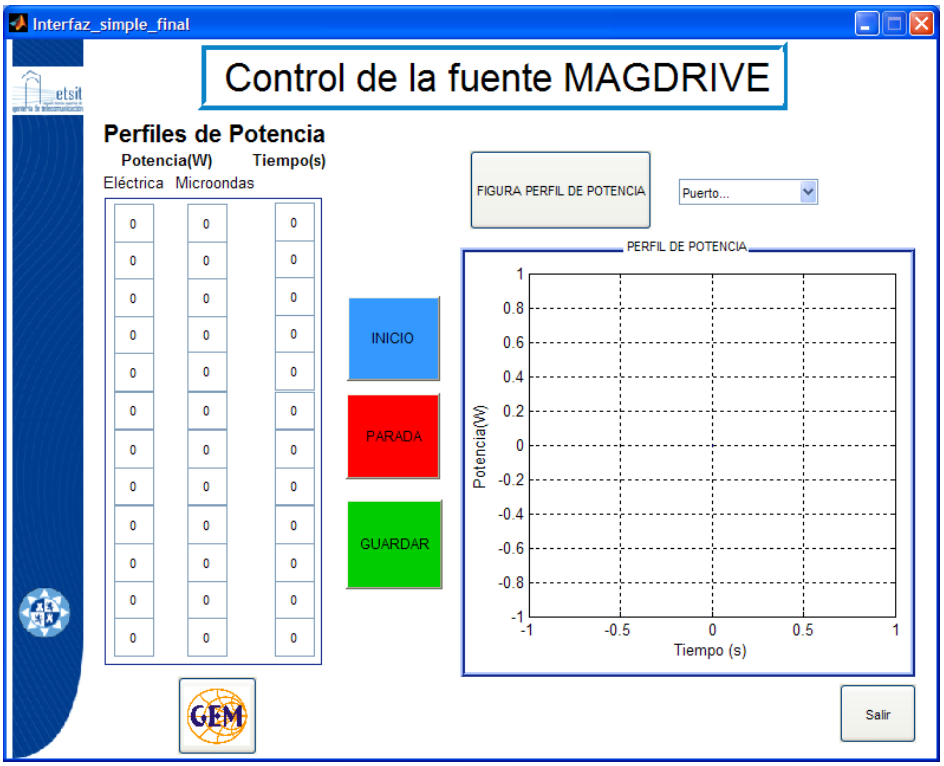

Figura 52: Interfaz Simple. (Windows)

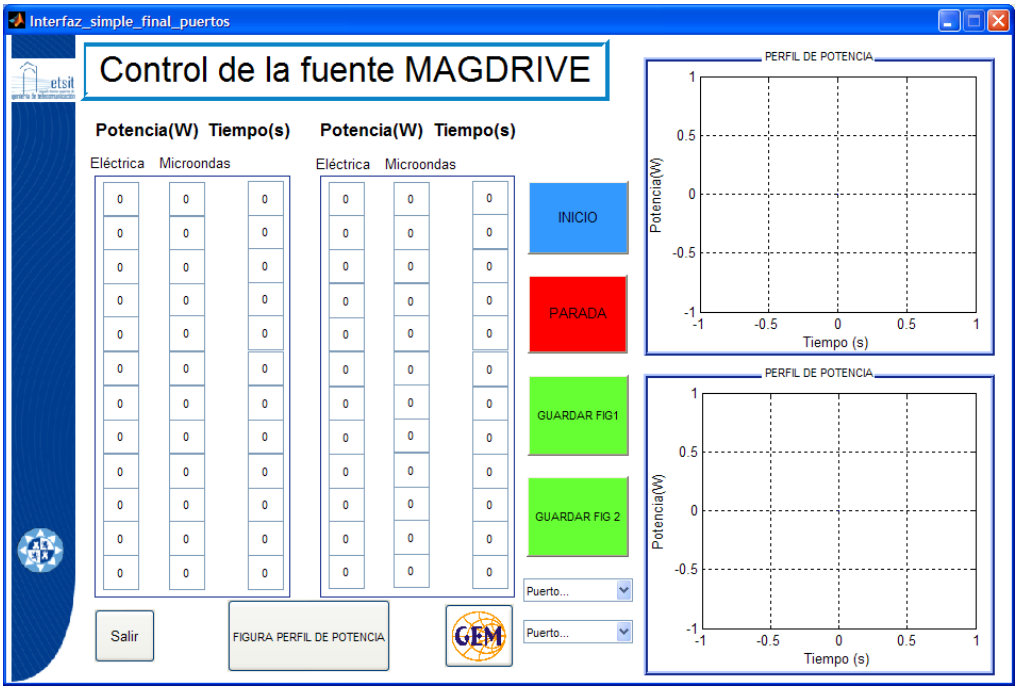

Figura 53: Interfaz Puertos. (Windows)

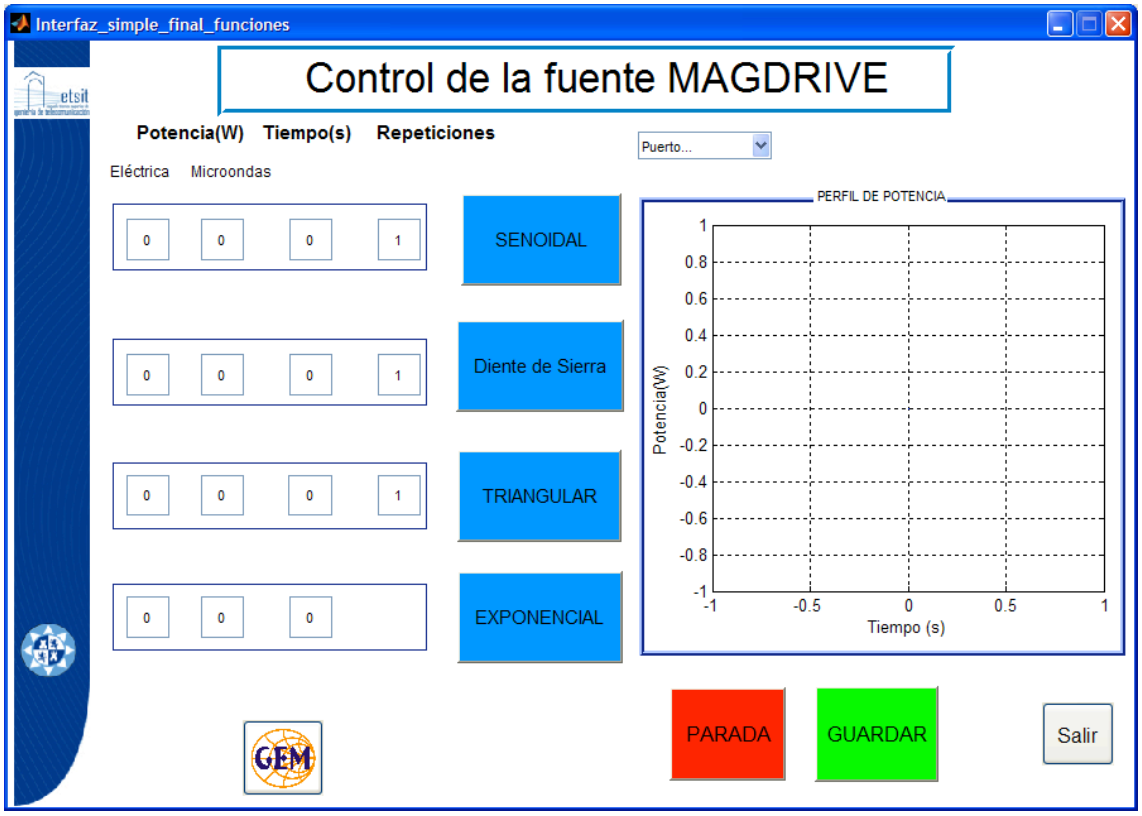

Figura 54: Interfaz Funciones. (Windows)

4. Montaje experimental.

4.1 Introducción.

A continuación, se van a exponer cada una de las pruebas realizadas en el laboratorio dónde se comprobó el funcionamiento de las interfaces. En primer lugar se va a describir el método seguido para realizar las pruebas y se analizarán los elementos más importantes. Posteriormente se mostrarán los resultados obtenidos en cada prueba y se analizarán los resultados obtenidos.

4.2 Elementos.

A continuación en este apartado vamos a analizar cada uno de los elementos usados para la comprobación del funcionamiento de las interfaces.

El sistema lo podemos dividir en dos partes. La primera de ellas es la que forman el ordenador y la fuente de alimentación MAGDRIVE1000 conectadas por un cable de comunicación RS232. La segunda parte está formada por un ventilador, un magnetrón Panasonic (2M244-M23), un circulador MUEGGE MW1003A-210EC, un sistema de refrigeración por agua y una cavidad en la que se introduce el material para calentar.

En las siguientes figuras se muestran en primer lugar, la fuente de alimentación MAGDRIVE1000 y en segundo lugar, la segunda parte del sistema.

Figura 55: Fuente de alimentación MAGDRIVE1000.

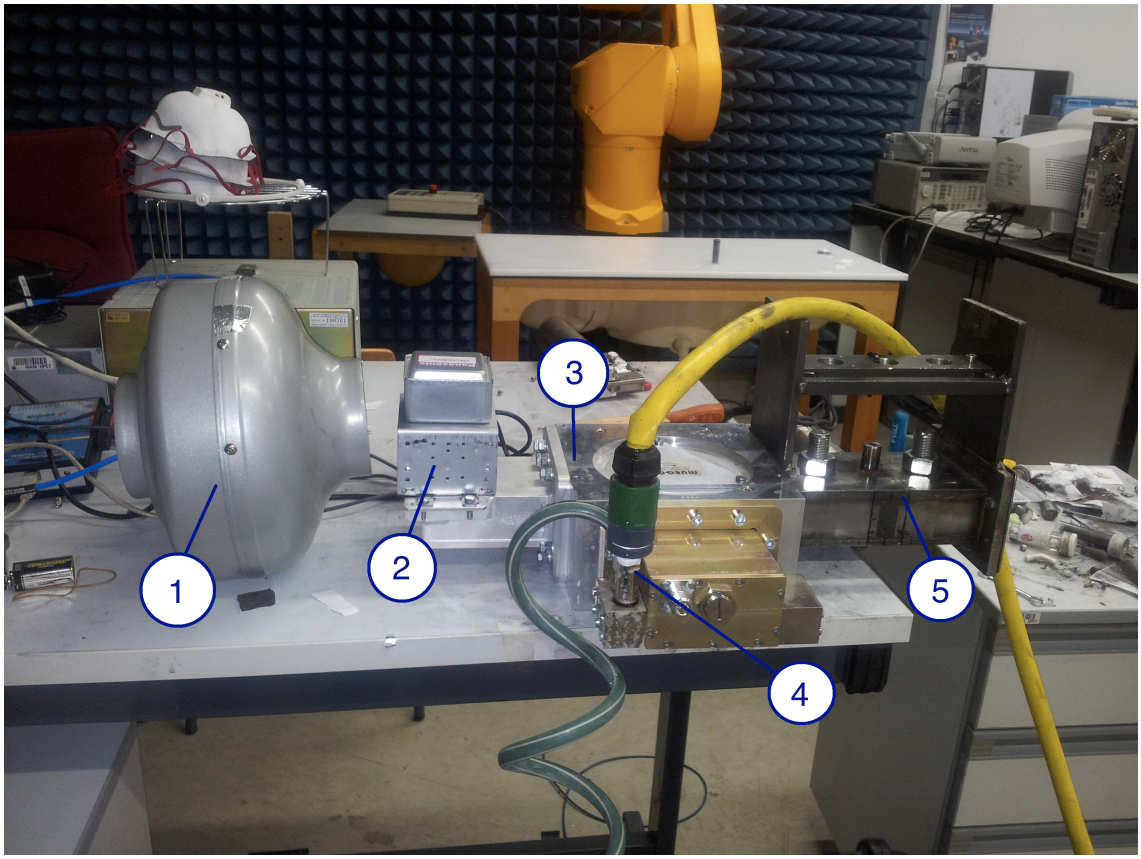

Figura 56: Segunda parte del sistema.

Los elementos enumerados son los siguientes:

1. Ventilador.
2. Magnetrón Panasonic (2M244-M23).
3. Circulador MUEGGE MW1003A-210EC.
4. Sistema de refrigeración por agua.
5. Cavidad.

4.2.1 Fuente de alimentación MAGDRIVE 1000.

Es el elemento más importante de todo el sistema, ya que es el que vamos a controlar con la interfaz. Su función principal será la de transmitir al magnetrón los patrones de potencia definidos por el usuario.

Además de transmitir la potencia al magnetrón, también se comunicará en todo momento con el ordenador verificando que la transmisión se está realizando sin errores. Por otro lado la temperatura dentro de la cavidad la mide un pirómetro añadido al montaje mostrado en la figura 56.

Datos eléctricos y medioambientales.

Datos eléctricos de entrada:

- Voltaje: 230(V) AC .
- Corriente: 6,5 A .
- Frecuencia: 50/60 Hz .
- Conexión: Cable $3\ x\ 1{,}5\ mm^2$, length 1m. Conector CEE - 7/XVII .

Datos eléctricos de salida:

- Potencia de salida: 100 – 1320(W) .
- Tensión nominal de ánodo: 4,3 (kV) (dependiente del magnetrón).
- Corriente nominal de filamento: 5-10 (A) .
- Magnetrón recomendado: Panasonic 2M244 .
- Conector: De dimensiones 6,3 x 0,8 mm.

Datos medioambientales:

- Caja: Caja metálica con ventilación forzada.
- Grado de protección: 1.
- Grado de contaminación: 2.
- Rango de temperaturas: 5-40ºC, 80% RH.

Medidas de la Fuente MAGDRIVE1000.

Las dimensiones y el peso de la fuente de alimentación son las siguientes:

- Peso: 2,5 (Kg).
- W x D x H: 165 x 230 x 130 mm.

En las siguientes figuras se muestran las medidas de una forma más detallada.

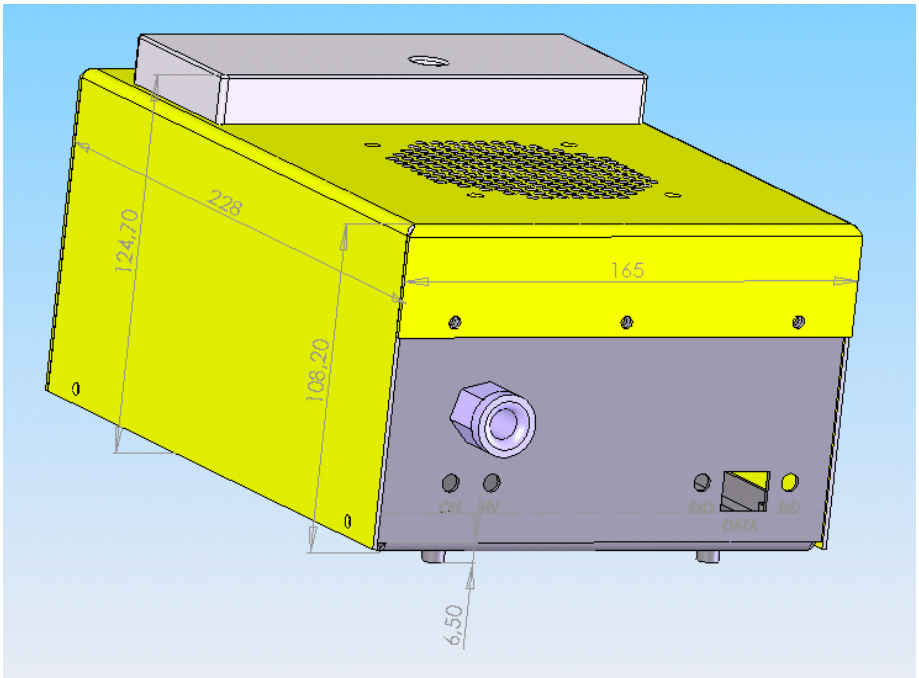

Figura 57: Dimensiones de la fuente MAGDRIVE1000.

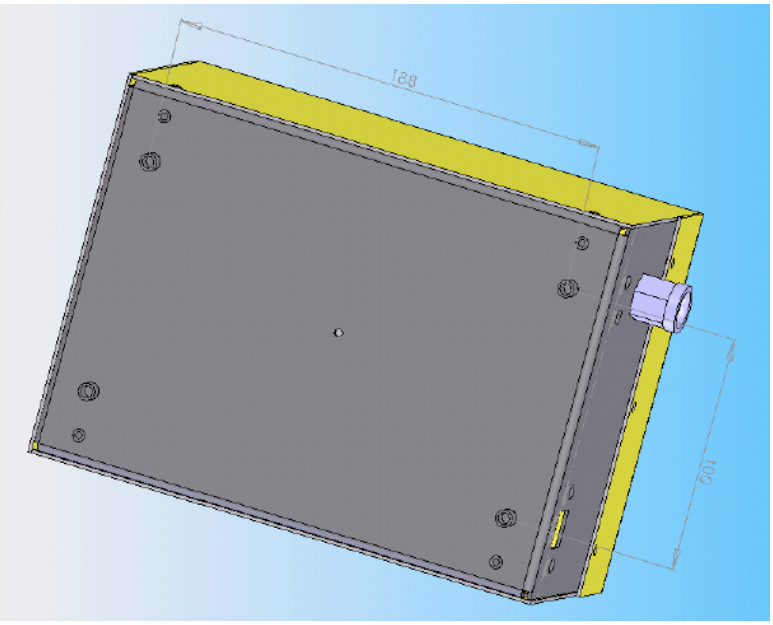

Figura 58: Huella de montaje de la fuente MAGDRIVE1000.

Indicadores.

Figura 59: Vista trasera de la fuente MAGDRIVE1000.

- ON: LED verde que indica 230 (V)AC y potencia interna.
- HV: LED rojo que indica Alta-Tensión de salida activa y/o filamento activo.
- RxD: LED amarillo que indica comunicación de un sistema de control externo.
- TxD: LED amarillo que indica comunicación de la fuente MAGDRIVE1000.

Cuando se conecta la fuente MAGDRIVE1000, el ventilador y el Led rojo HV se activan durante unos pocos segundos de prueba para una prueba de verificación del funcionamiento. No se produce alto voltaje durante la prueba de verificación.

Comunicación.

Se usa un cable RS-232 modificado, usando las señales RxD, TxD, RTS y DTR conectadas a un conector de 8 pines.

La configuración de la comunicación será la siguiente: Velocidad de transmisión 2400, 8 bits de datos, 1 bit de parada, sin paridad, la señal RTS (Request to send) siempre en estado alto y la señal DTR (Data terminal ready) siempre en estado bajo.

La comunicación puede paralelizarse, lo que permite controlar hasta tres fuentes MAGDRIVE1000 con un puerto serie.

A cada unidad se le puede asignar una dirección de 1 a 255. La dirección '0' es una dirección *broadcast* que todas la unidades pueden escuchar.

El cable de comunicación puedes construirse de manera sencilla usando conectores estándar de la siguiente forma.

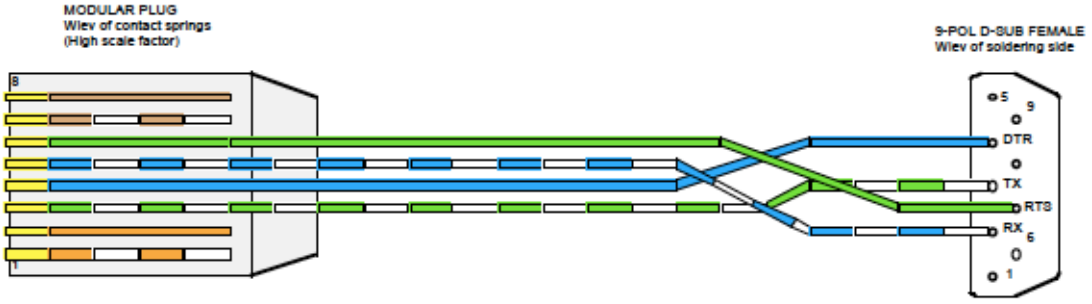

Figura 60: Comunicación con la fuente MAGDRIVE1000.

Protocolos de comunicación:

Comandos de escritura.

Envío de datos					Datos recibidos de MAGDRIVE1000			
Comandos	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Byte 3
Apagado	128	37	0	165	37	0	37	Off Byte 2 =0→7
Potencia 80W	128	37	8	173	37	8	45	
Potencia 90W	128	37	9	174	37	9	46	
Potencia 100W	128	37	10	175	37	10	47	
Potencia 200W	128	37	20	185	37	20	57	
Potencia 300W	128	37	30	195	37	30	67	
Potencia 400W	128	37	40	205	37	40	77	
Potencia 500W	128	37	50	215	37	50	87	
Potencia 800W	128	37	80	245	37	80	117	
Potencia 1KW	128	37	100	265	37	100	137	
Potencia 1,3KW	128	37	130	295	37	130	167	

Figura 61: Comandos de escritura MAGDRIVE1000.

- Byte 2 = P(W)/10
- Byte 3 = Byte 0 + Byte 1 + Byte 2
- Byte 2 > 130 = Denegado por la fuente MAGDRIVE1000
- Byte 2 < 8 = Apagado de la fuente

Comandos de lectura.

XX: Solicitud del valor devuelto por la fuente MAGDRIVE1000.

YY: Checksum = Byte 0 + Byte 1

Envío de datos					Datos recibidos de MAGDRIVE1000			
Comandos	Byte 0	Byte 1	Byte 2	Byte 3	Byte 0	Byte 1	Byte 2	Comentario
Leer Potencia	0	37	0	37	37	XX	YY	Devuelve valor potencia
Leer Alarma	64	24	0	88	24	XX	YY	Devuelve estado Alarma
Leer Tiempo	0	43	0	43	43	XX	YY	Devuelve Temperatura
Leer Corriente	0	59	0	59	59	XX	YY	Devuelve corriente x10

Figura 62: Comandos de lectura MAGDRIVE1000.

Registro de Alarmas.

ALARMAS	
Byte 0	'Fallo, control del Filamento ','GEM'
Byte 1	'Fallo, control de Potencia Anódica ','GEM'
Byte 2	'Fallo, control de Temperatura ','GEM'
Byte 3	'N.C','GEM'
Byte 4	'Regulación de Temperatura en progreso ','GEM'
Byte 5	'Fallo, control PFC','GEM'
Byte 6	'N.C','GEM'
Byte 7	'Unidad deshabilitada debido a una alarma ','GEM'

Figura 63: Registro de alarmas MAGDRIVE1000.

Funcionamiento eléctrico de la comunicación de datos.

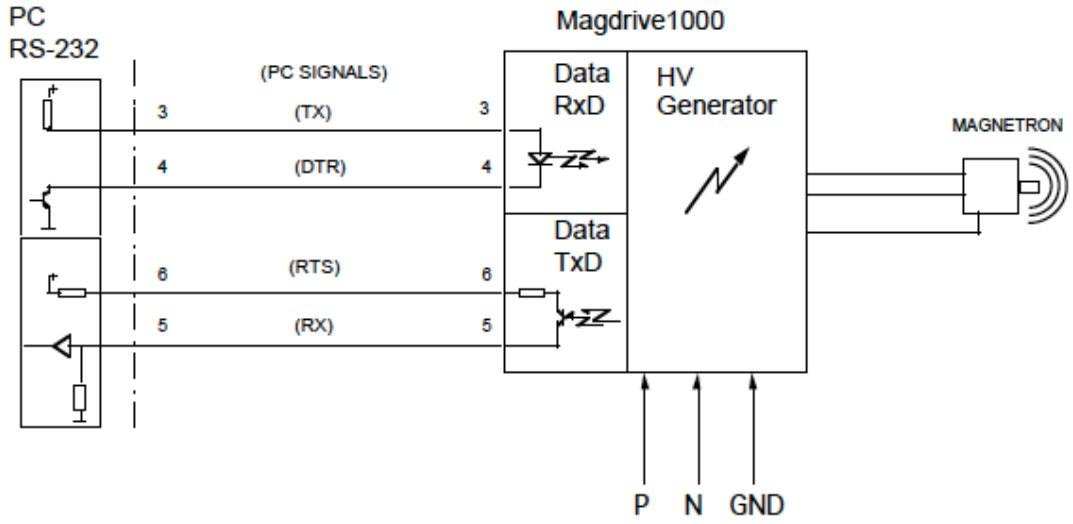

Figura 64: Esquema del funcionamiento eléctrico (Comunicación).

La señal TX envía información del ordenador a la fuente MAGDRIVE1000. El lazo de corriente vuelve al PC a través de la señal DTR que debe encontrarse en baja.

La señal RTS establecida en alto permite a la fuente MAGDRIVE1000 enviar señales a través del opto acoplador al PC a través de la señal RX.

Conexión eléctrica.

Conexión de encendido y apagado.

La fuente MAGDRIVE1000 se conecta a una toma de corriente estándar, no tiene un interruptor de apagado y encendido, se debe estar seguro que la alimentación se encuentra desconectada antes de mover o desconectar la fuente. Se debe dejar al menos un minuto para que los condensadores se descarguen.

Si la fuente MAGDRIVE1000 se encuentra permanente conectada a terminales fijos, debe asegurarse que existe algún interruptor o dispositivo usado para desconectar la unidad.

Fusible interno.

Un fusible interno de 10 A de acción lenta, protege la línea de alimentación de la fuente MAGDRIVE1000 de fallos de funcionamiento.

Fusible interno: Cerámico 5x20 mm. 10A. Alta capacidad de rotura (1500A).

Fusible externo.

La potencia de microondas radiada por el magnetrón recomendado Panasonic 2M244 es de un KW. Esto requiere 1300W que deben ser entregados de la fuente MAGDRIVE1000. El fusible externo es capaz de 8-10ª.

Recomendación de seguridad.

La fuente MAGDRIVE1000 debe siempre estar conectada a una toma de tierra.

Cableado de alto voltaje.

Tres cables de alto voltaje son usados para conectar la fuente al magnetrón. Dos cables llevan corriente al magnetrón y uno lleva la corriente anódica de retorno.

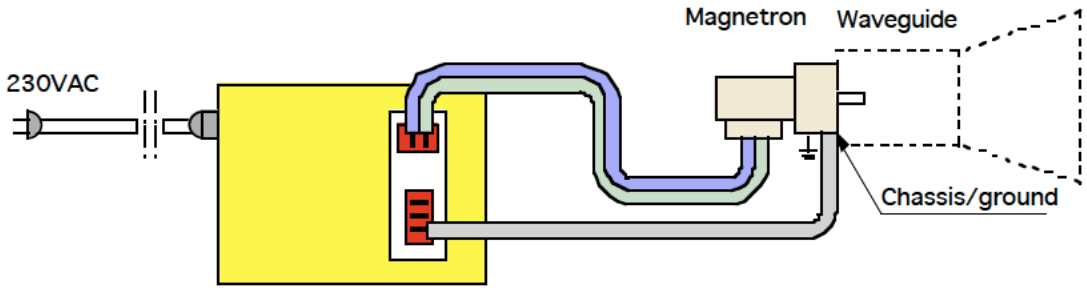

Figura 65: Esquema fuente MAGDRIVE1000 y magnetrón.

4.2.2 Magnetrón Panasonic (2M244-M23).

El magnetrón usado es el recomendado por el fabricante de la fuente MAGDRIVE1000, ya que cumple todas las características y proporciona el mejor rendimiento a la fuente.

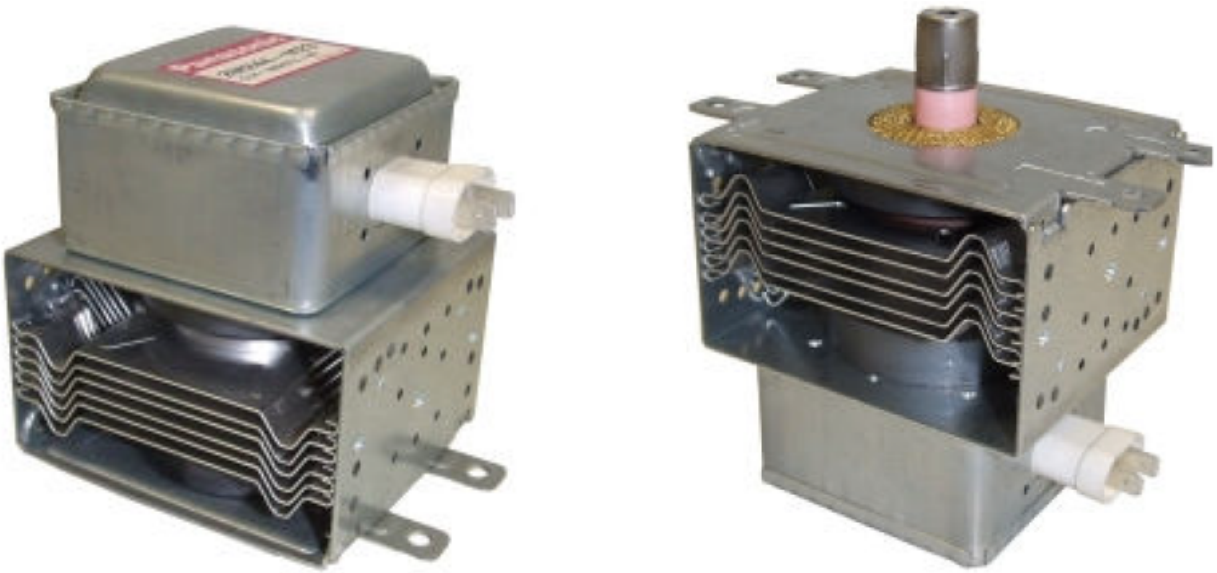

Figura 66: Magnetrón Panasonic .

Especificaciones.

Características	Especificaciones
Frecuencia de operación	2470 MHz
CW Potencia de salida	1060 W
Voltaje ánodo	4,55 KV
Corriente ánodo	370 mA
Voltaje filamento	3,5 V
Corriente filamento	12 A
Transmisión de RF	En una guía onda rectangular o en una cavidad directamente.
Eficiencia	Aproximadamente 72%
Refrigeración por aire	Aire forcado (>1000 l/min)
Peso neto	Aproximadamente 0.9 Kg

Figura 67: Especificaciones Magnetrón Panasonic .

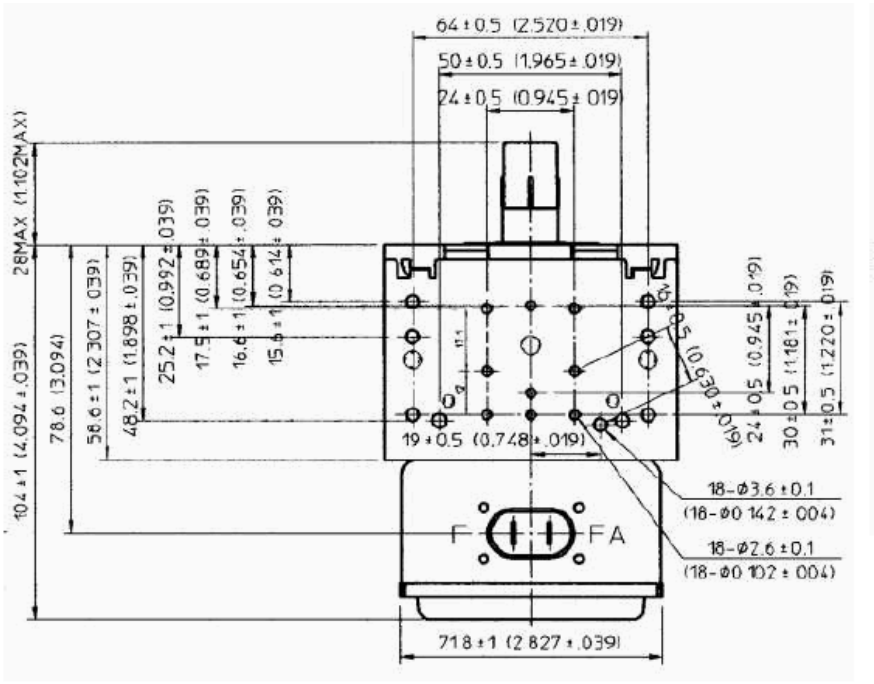

Figura 68: Dimensiones Magnetrón Panasonic.

En la siguiente figura se puede apreciar el montaje del magnetrón en el sistema realizado para las pruebas en el laboratorio.

Figura 69: Montaje Magnetrón Panasonic.

4.2.3 Circulador Muegge MW1003A-21EC.

El circulador usado para las pruebas en el laboratorio es el siguiente:

Figura 70: Circulador Muegge.

Especificaciones.

Características	Especificaciones
Frecuencia de operación	2,470 MHz ± 25 MHz
Pérdidas insertadas	0,2 dB, max
Aislamiento	23 dB, min
Potencia Transmitida	3 W
Potencia Reflejada	3 W
Rango de Temperaturas	10 - + 50 ºC
Enfriamiento	+40ºC máx (Agua interna) Ay 18 – 20ºC 3L/min, 4 bar máx
Dimensiones	182 mm (W) x 300mm(D)x 98mm(H)

Figura 71: Especificaciones circulador Muegge.

Figura 72: Circulador Muegge ensamblado en el montaje.

5. Resultados

5.1 Introducción.

En esta sección se va a proceder a la exposición de las medidas y resultados de las pruebas realizadas en el laboratorio usando las interfaces ya explicadas en los apartados anteriores.

En cada una de las pruebas que se mostrarán en los siguientes apartados, se observará en primer lugar la interfaz usada una vez que han sido introducidos los datos que constituirán el perfil de potencia y a continuación una gráfica con los resultados obtenidos. Dichas gráficas muestran la temperatura que ha alcanzado el material dentro de la cavidad el cual es un trozo de Silar® , que es un tipo de cerámica que aguanta altas temperaturas. a lo largo de todo el proceso de medición.

Las medidas obtenidas del pirómetro en cada una de las pruebas se usan para la exposición de los resultados. Al guardar los resultados del proceso de calentamiento, el software del pirómetro genera un archivo con todos los datos medidos durante la prueba, siendo sólo necesarios, el tiempo y la temperatura del objeto que se corresponden con las dos primeras columnas. En la siguiente captura se puede ver una pequeña parte del contenido del archivo generado.

```
[Connect DataFile][1.2]
Date:  12/15/2011
Time:  09:48:54
Unit:  ∞C
Resolution:  0.001/0.100
Values:      7

Time    TObj   TInt   TBox   TAct   T2C    T1C    ATTENUATION

00:00:00.000 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.001 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.002 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.003 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.004 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.005 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.006 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.007 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.008 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.009 46.6   20.0   23.7   46.6   0.0    0.0    0.0
00:00:00.010 46.6   20.0   23.7   46.6   0.0    0.0    0.0
```

Figura 73: Toma de medidas.

Se debe tener en cuenta que en las gráficas de resultados el eje de ordenadas encontramos la temperatura (ºC) y en el de abscisas el tiempo (s) el cual se encuentra en el siguiente formato `00:00:00.000`. Debido a esto, se obtienen esa gran cantidad de datos para el tiempo establecido.

5.2 Prueba 1.

La prueba número 1 se ha realizado con la interfaz simple. En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

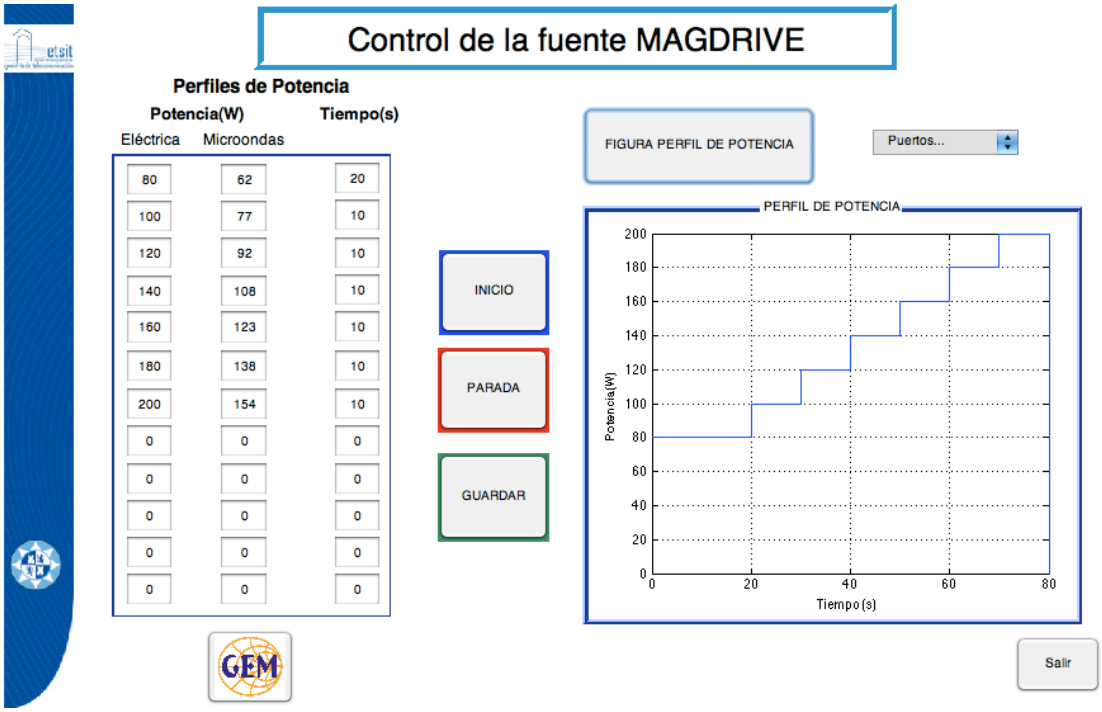

Figura 74: Interfaz simple Prueba 1.

La siguiente gráfica muestra la medida obtenida al calentar el objeto con el perfil de potencia introducido.

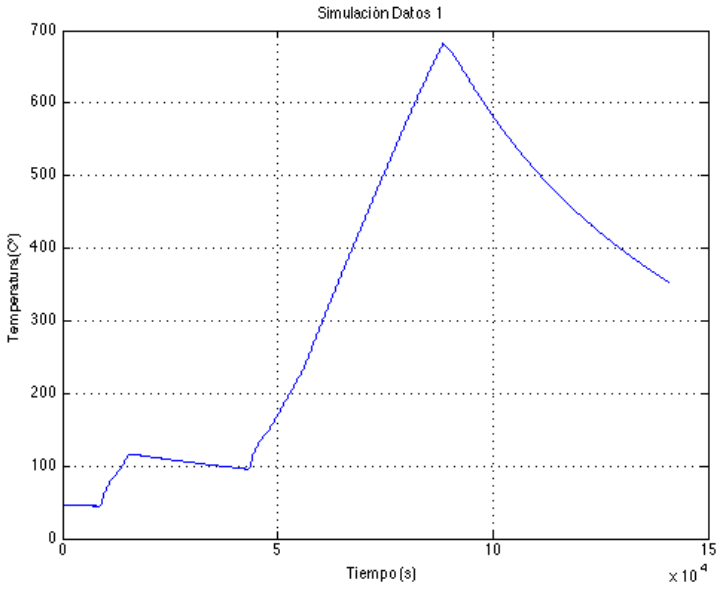

Figura 75: Resultado Prueba 1.

Como puede observarse en la gráfica, al inicio de ponerse en funcionamiento el sistema, la temperatura del objeto aumenta alcanzando 115.7ºC y descendiendo posteriormente adaptándose así a los valores de potencia introducidos en la interfaz. Este aumento de temperatura se debe a que la fuente de alimentación al ponerse en funcionamiento siempre transmitirá un pico de potencia calentando en todas las pruebas realizadas el material al inicio.

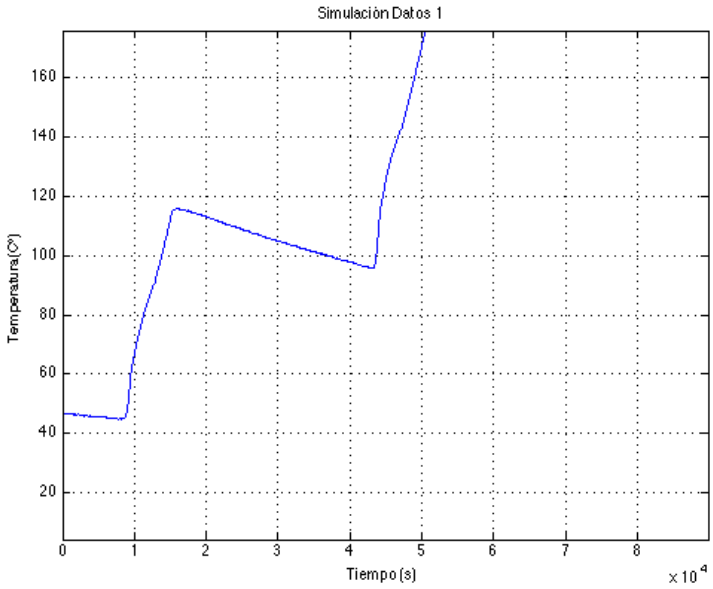

Figura 76: Detalle (1) Resultado Prueba 1.

Una vez que ha descendido la temperatura del material, se puede apreciar la evolución del perfil de potencia. Se están transmitiendo 80W de durante 20 segundos calentando la muestra alrededor de los 100 grados.

Justo después, se puede comprobar como cambia la pendiente, porque la fuente comienza a mandar 100W de potencia durante 10 segundos. Como cada 10 segundo la fuente va aumentando la potencia de una manera escalonada, se puede ver como la temperatura aumenta manteniendo en casi todo momento la misma pendiente.

Debido a que la muestra está siendo calentada de forma continua durante 80 segundos, ésta llega a alcanzar una temperatura 682,5ºC

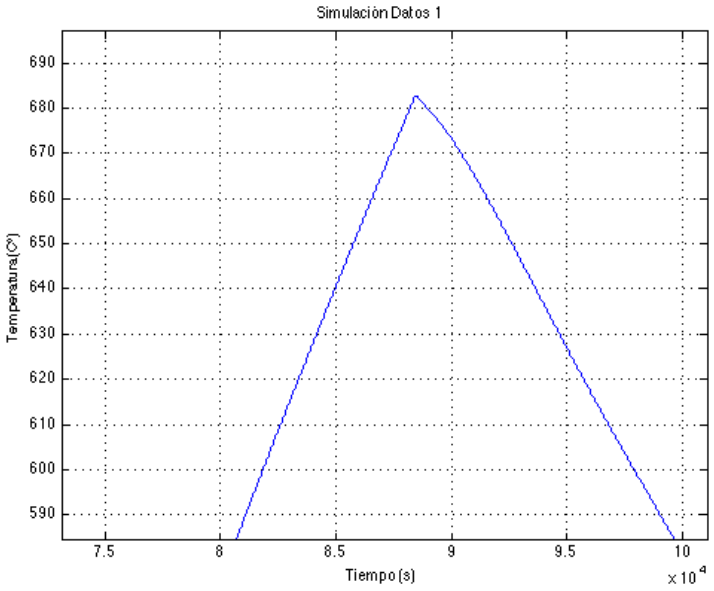

Figura 77: Detalle (2) Resultado Prueba 1.

Cuando han transcurrido 80 segundos, la fuente deja de transmitir, ya que al final de la transmisión del patrón de potencias, siempre se envía un comando de apagado y se cierra la comunicación. Este efecto se puede comprobar en este último tramo de la gráfica, donde se puede ver como la temperatura desciende tras alcanzar el valor máximo antes del apagado de la fuente.

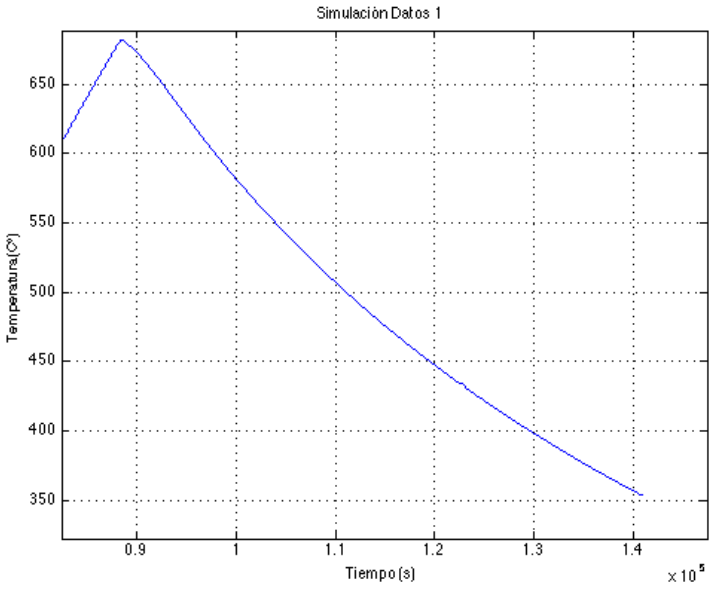

Figura 78: Detalle (3) Resultado Prueba 1.

5.3 Prueba 2.

La prueba número 2 también se ha realizado con la interfaz simple. En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

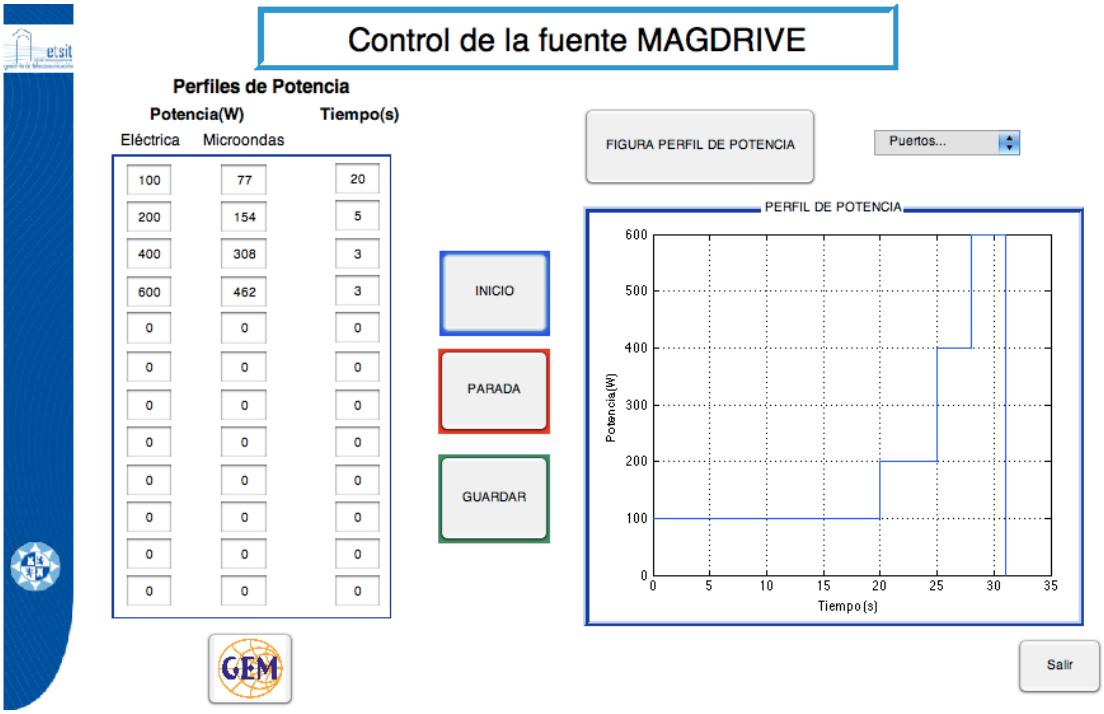

Figura 79: Interfaz simple Prueba 2.

El resultado obtenido es el siguiente:

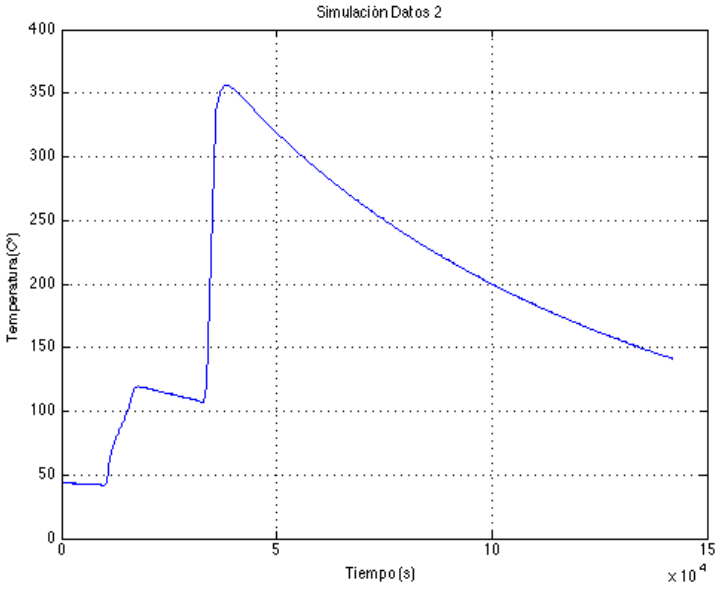

Figura 80: Resultado Prueba 2.

Como ocurría en la anterior prueba, debido al pico de potencia que se transmite al principio se tiene el mismo efecto de calentamiento. Una vez que desciende la temperatura se aprecia el efecto de la potencia de 100W transmitida durante 20 segundos.

Transcurrido este tiempo, la fuente comienza a transmitir las potencias de 200W 400W y 600W. Como consecuencia de este aumento de potencia, el magnetrón calentará la muestra y la temperatura medida aumenta con una pendiente elevada tal y como se muestra en la siguiente figura.

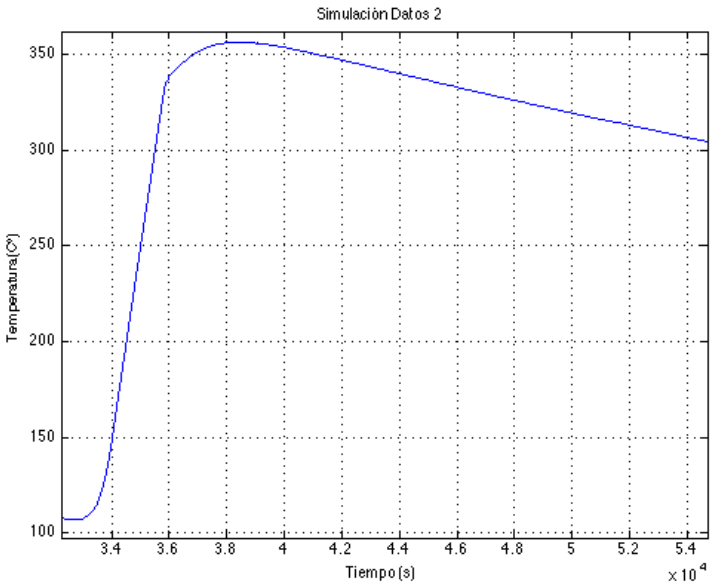

Figura 81: Detalle Resultado Prueba 2.

Cuando han transcurrido 31 segundos y la transmisión de potencia se ha completado se envía un comando de apagado a la fuente y se cierra la comunicación.

5.4 Prueba 3.

La prueba número 3 se ha realizado con la interfaz simple. En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

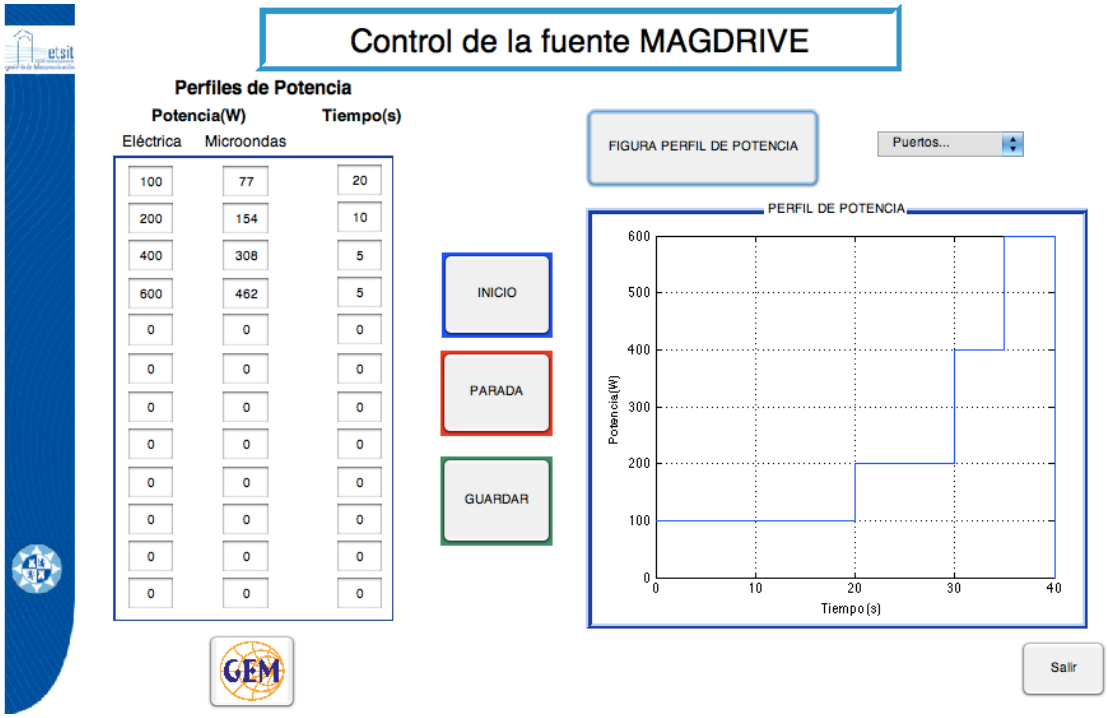

Figura 82: Interfaz Simple Prueba 3.

El resultado obtenido es el siguiente:

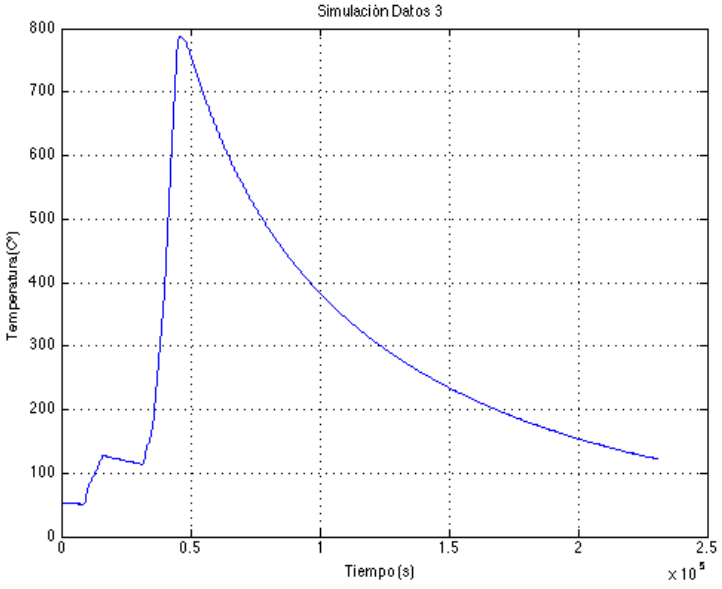

Figura 83: Resultado Prueba 3.

Como en el resto de resultados obtenidos, se inicia con un calentamiento de la muestra debido al pico de potencia inicial con un posterior descenso de temperatura. El resultado obtenido en esta prueba es muy similar al de la prueba 2, pero en este caso se puede ver como la potencia máxima alcanzada es mucho más elevada llegando hasta los 787ºC.

El aumento de temperatura es debido a que se han duplicado los tiempos de transmisión, de este modo se transmitirán 200W durante 10 segundos , 400W durante 5 segundos y 600W durante 5 segundos.

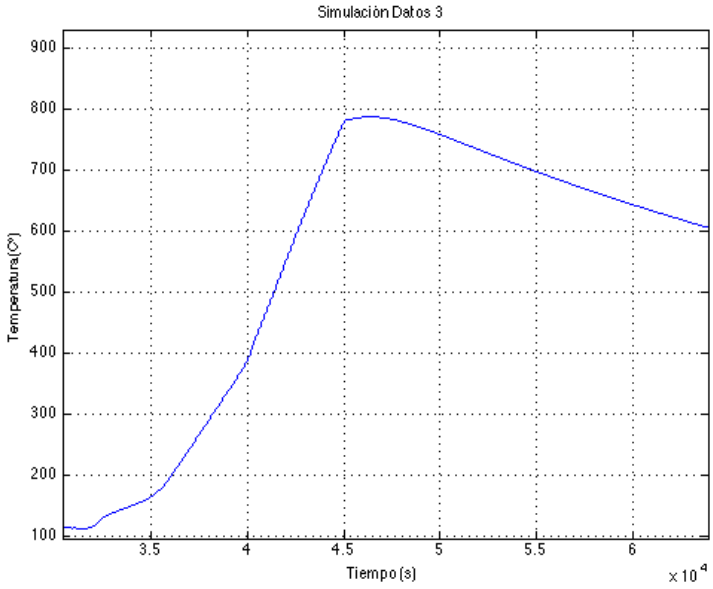

Figura 84:Detalle Resultado Prueba 3.

A los 40 segundos la fuente deja de transmitir y la temperatura disminuye.

5.5 Prueba 4.

Para la prueba número 4 se ha usado la interfaz de funciones. De los patrones de potencia que se pueden elegir, en este caso se ha decidido seguir una distribución senoidal.

En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

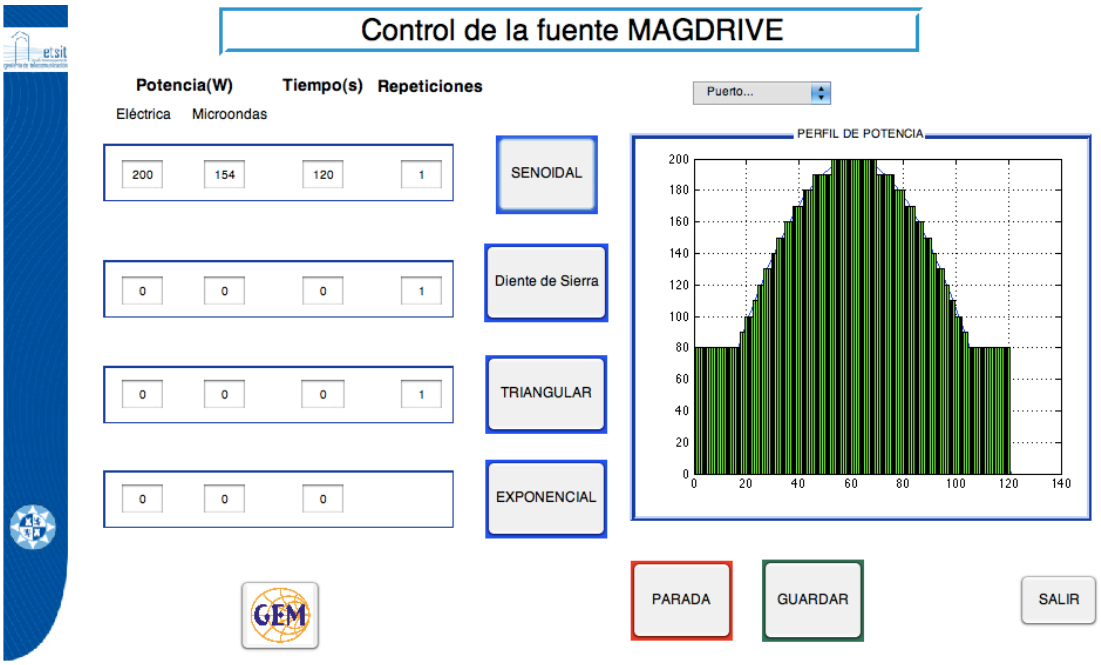

Figura 85: Interfaz Funciones Prueba 4.

El resultado obtenido es el siguiente:

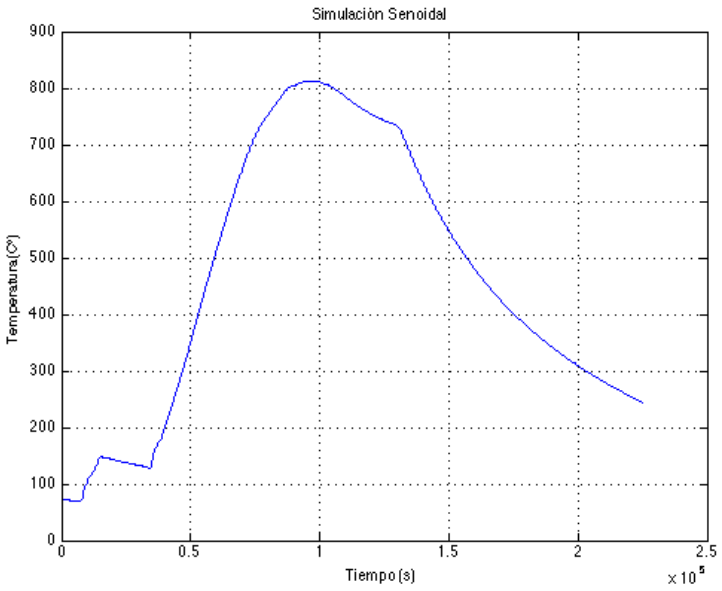

Figura 86: Resultado Prueba 4.

Se puede apreciar en primer lugar el calentamiento debido al pico de tensión inicial. A continuación, se comprueba como se produce un aumento de la temperatura a los 20 segundos, que es cuando la potencia emitida por la fuente comienza a crecer siguiendo la distribución senoidal.

Como se comentó en apartados anteriores aparte del límite inferior de 80W y el límite superior de 1300W, los valores de potencia se redondean en decenas. Debido a este redondeo, cuando se llega al valor máximo del seno, la potencia de 200W se mantiene durante varios segundos, tal y como puede verse en la representación del perfil de potencia.

A medida que aumenta la potencia radiada, la temperatura del objeto se va elevando hasta alcanzar un máximo de 812,5 ºC. Si se observa detenidamente el valor máximo de temperatura alcanzado, no corresponde con 60 segundos tal y como debería ser, sino que el valor máximo se alcanza a los 97 segundos cuando la potencia transmitida por la fuente es de 90W. El motivo de esta variación se debe a que el material no puede enfriarse con tanta rapidez, ya que sigue calentándose al recibir el calor de las microondas, que aunque sea cada vez menor hasta que no disminuye lo suficiente el material no puede comenzar a enfriarse. Una vez que el calor transmitido al cuerpo y la temperatura del cuerpo son adecuados, el cuerpo comienza a enfriarse.

Finalmente a los 120 segundos la fuente deja de transmitir y el cuerpo se va enfriando con una mayor rapidez.

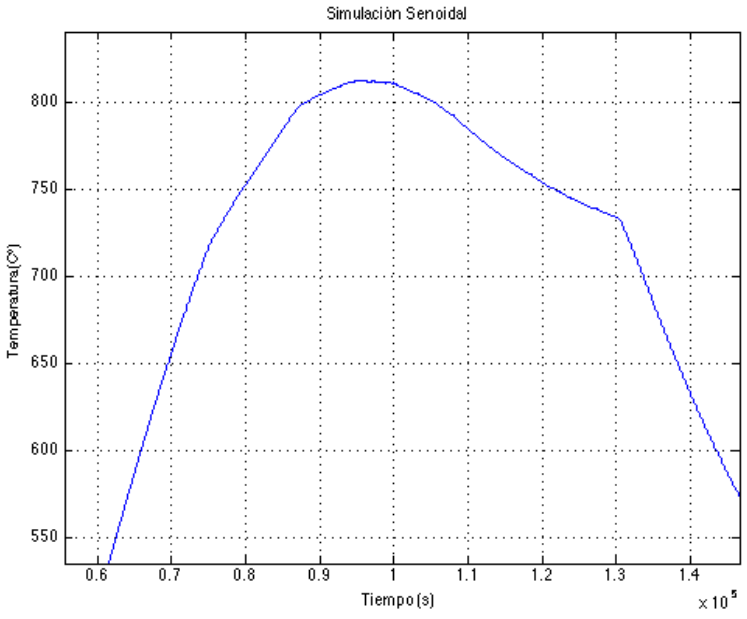

Figura 87: Detalle Resultado Prueba 4.

5.6 Prueba 5.

Para la prueba número 5 se ha usado la interfaz de funciones. De los patrones de potencia se ha decidido seguir una distribución diente de sierra.

En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

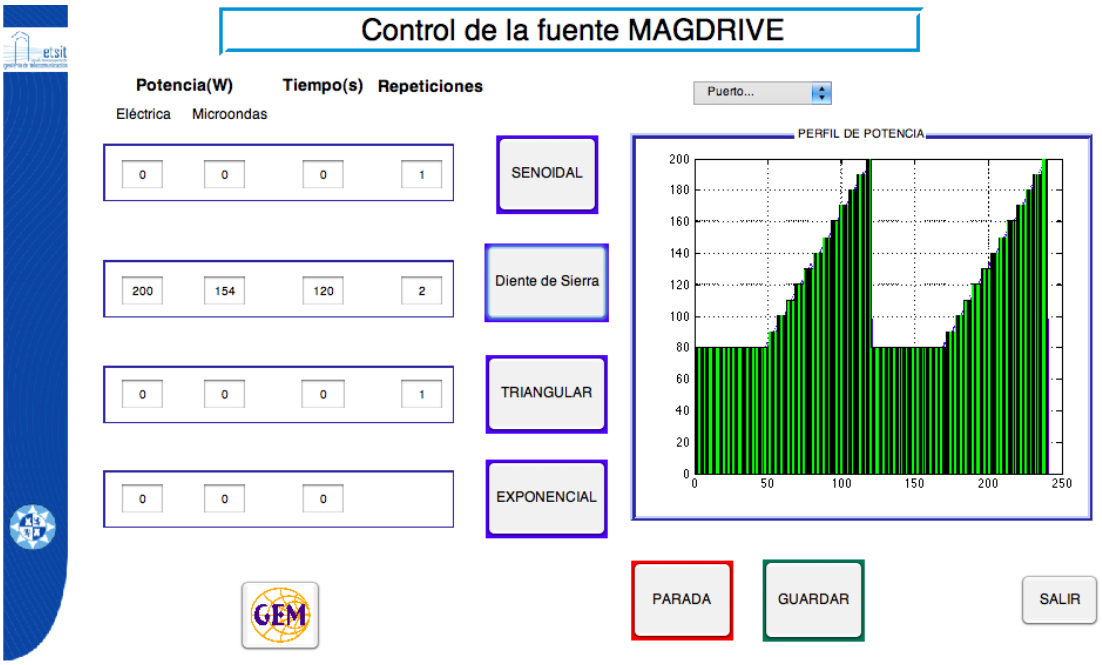

Figura 88: Interfaz Funciones Prueba 5.

El resultado obtenido es el siguiente:

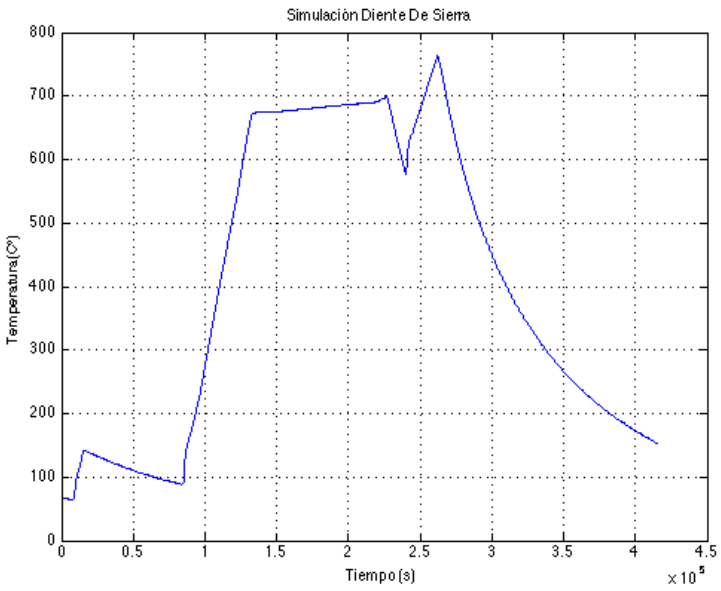

Figura 89: Resultado Prueba 5.

Se puede apreciar en primer lugar el calentamiento debido al pico de tensión inicial. A continuación, se comprueba como se produce un aumento de la temperatura entorno a los 60 segundos, que es cuando la potencia emitida por la fuente comienza a aumentar siguiendo la distribución diente de sierra.

Una vez que comienza a aumentar la temperatura, se llega a una máxima de 674ºC a los 144 segundos, no a los 120 segundos encontrándose el material a una temperatura de 526,2ºC, que se corresponde con el valor de potencia máximo transmitido si se observa el perfil de potencia. A partir de ese momento, la pendiente se atenúa y el aumento de temperatura es cada vez menor, hasta llegar a un máximo final de 699ºC a los 226 segundos.

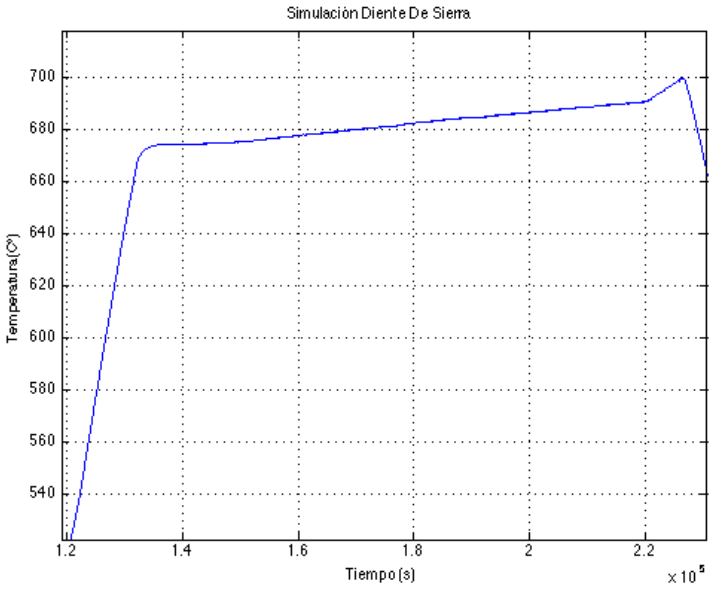

Figura 90: Detalle (1) Resultado Prueba 5.

Esta no correspondencia entre la temperatura medida y el perfil de potencia, tiene la misma explicación que en la prueba anterior, en la que se aplicó un perfil que seguía una distribución senoidal.

El material sigue calentándose durante aproximadamente unos 20 segundos hasta que se estabiliza, alcanzando un máximo a los 226 segundos. Entonces comienza a descender la temperatura hasta un mínimo de 577,8ºC a los 240 segundos, que se corresponde con el segundo pulso del diente de sierra donde la potencia transmitida son 140W.

En este momento, la temperatura del objeto no debería de descender, sino que debería aumentar. La explicación de este error se ha asumido tras la realización de varias pruebas que se han descartado para poder discernir el motivo de este error. Esto es debido a que cuando la fuente cambia tan rápidamente de valores de potencia y se encuentra entre valores de un intervalo comprendido entre 100W – 350W, comete errores en la transmisión y tarda unos segundos en volver a enviar los valores correctos.

En este caso la fuente siguió enviando valores de potencia de 80W y cuando la fuente se estabilizó, se volvieron a enviar los valores correctos. Por eso la temperatura descendía en ese tramo cuando no debía hacerlo. En la siguiente figura se pueden apreciar estos errores.

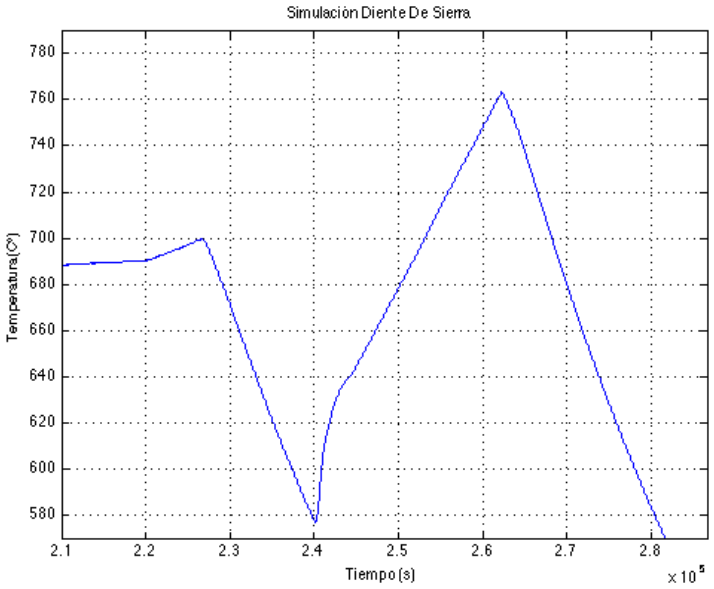

Figura 91: Detalle (2) Resultado Prueba 5.

Finalmente a los 260 segundos se alcanza la temperatura máxima de 763,4ºC, no a los 240 segundos que es cuando la fuente deja de funcionar, lo cual se debió al calor que se ha mantenido en la cavidad. En las demás pruebas realizadas al ser de menor duración el aumento de temperatura tras la desconexión, se consideró un hecho trivial, sin embargo en este caso se ha considerado relevante y se ha expuesto porque este tiempo ha sido bastante mayor.

5.7 Prueba 6.

Para la prueba número 6 se ha usado la interfaz de funciones. De los patrones de potencia se ha decidido seguir una distribución triangular.

En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

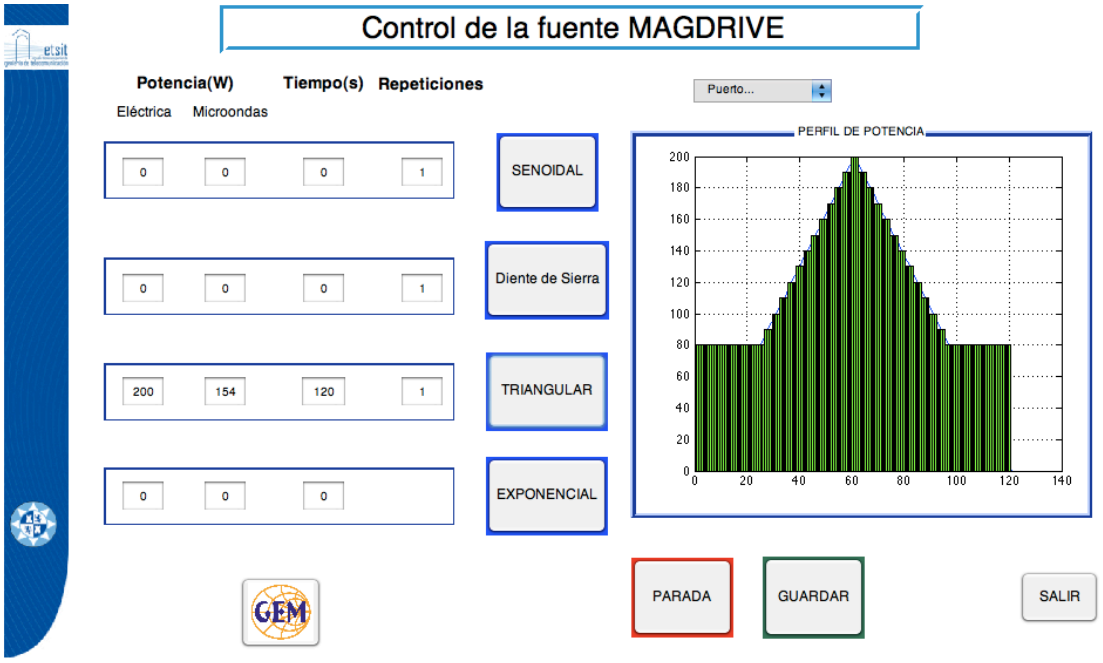

Figura 92: Interfaz Funciones Prueba 6.

El resultado obtenido es el siguiente:

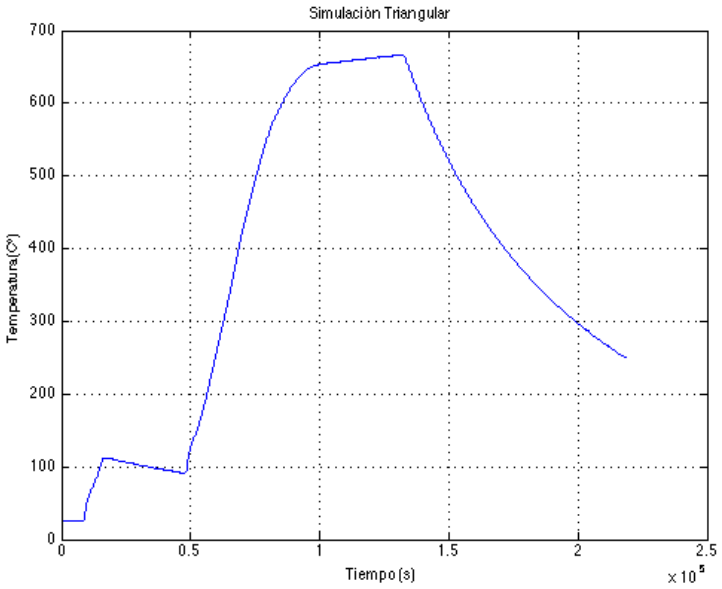

Figura 93: Resultado Prueba 6.

Como siempre, al inicio se puede apreciar el calentamiento por el pico de tensión. Si nos fijamos, el resultado obtenido es muy similar al de la simulación senoidal. Sin embargo, debido a que se sigue una distribución triangular, la pendiente es mucho más pronunciada en este caso.

En la siguiente captura se puede ver una comparación entre la distribución senoidal y triangular.

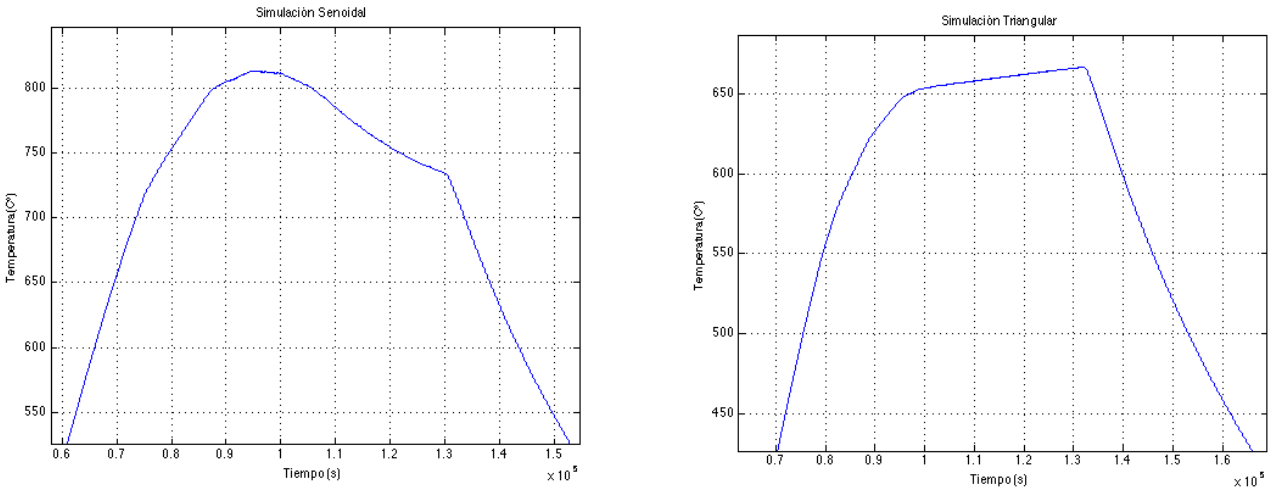

Figura 94: Detalle Resultado Prueba 6.

Esta diferencia entre ambos resultados en el intervalo de temperatura máxima, es la que se esperaba, por el contraste de las pendientes entre los perfiles de potencia.

Además si nos fijamos en la respuesta triangular vuelve a ocurrir el mismo error que en la prueba senoidal donde la temperatura de la muestra tarda en descender cuando los valores de potencia se corresponden con la pendiente negativa. La temperatura de la muestra no ha llegado a descender como en la senoidal, sino que en este caso sólo se ha aplanado la pendiente.

Finalmente una vez terminada la transmisión, se apaga la fuente y se cierra la comunicación.

5.8 Prueba 7.

Para la prueba número 7 se ha usado la interfaz de funciones. De los patrones de potencia se ha decidido seguir una distribución exponencial.

En la siguiente figura se pueden observar los valores que se han introducido que constituyen el perfil de potencia enviado a la fuente MAGDRIVE1000.

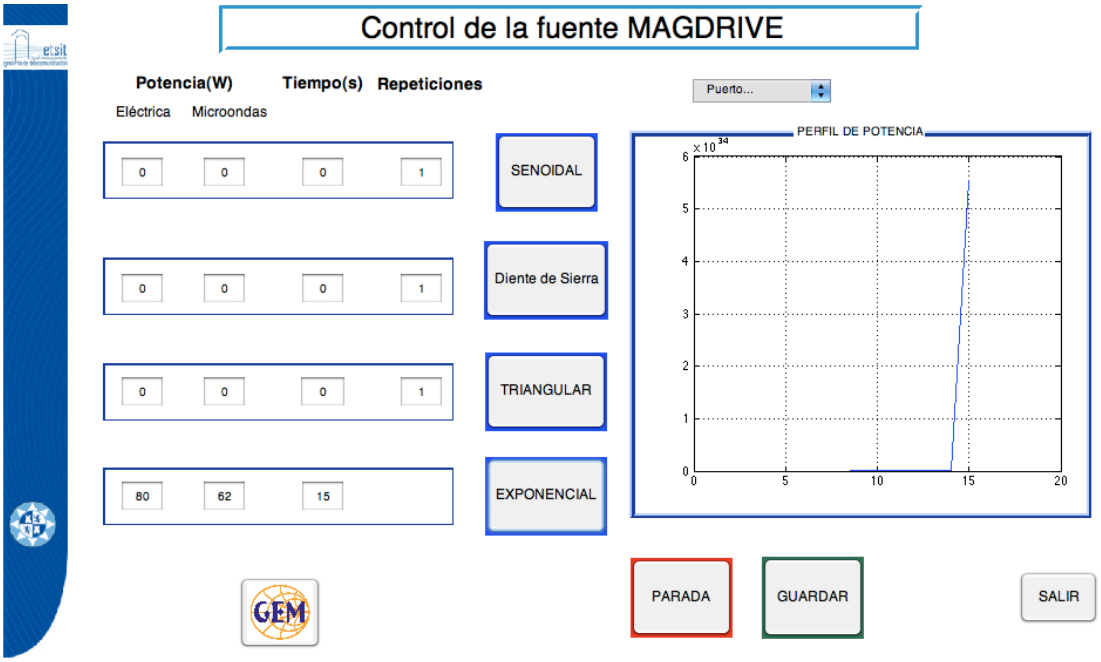

Figura 95: Interfaz Funciones Prueba 7.

El resultado obtenido es el siguiente:

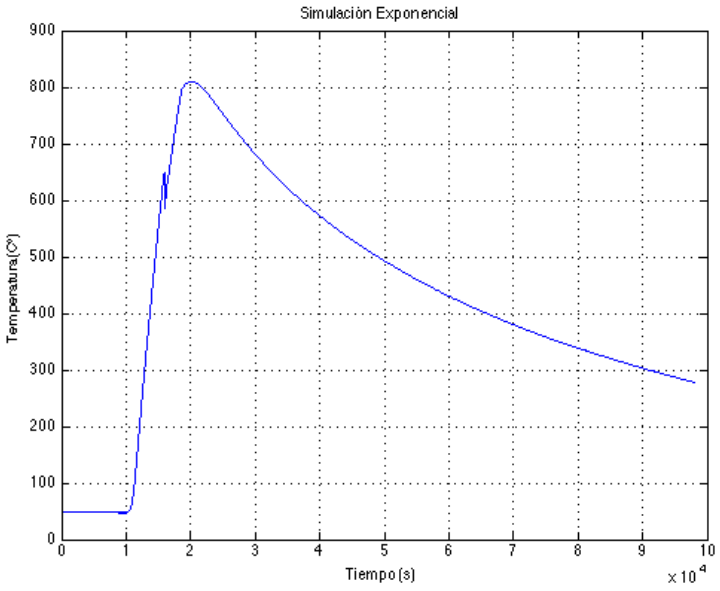

Figura 96: Resultado Prueba 7.

Analizando el resultado se puede ver que en primer lugar el calentamiento de la muestra debido al pico de tensión inicial.

Cuando se sigue esta distribución, los valores se verán truncados rápidamente, ya que la fuente sólo puede transmitir una potencia máxima de 1300W. En las siguientes capturas del perfil de potencia se ha hecho un zoom para poder mostrar el perfil de potencia que se va a transmitir sitiado en la imagen de la derecha.

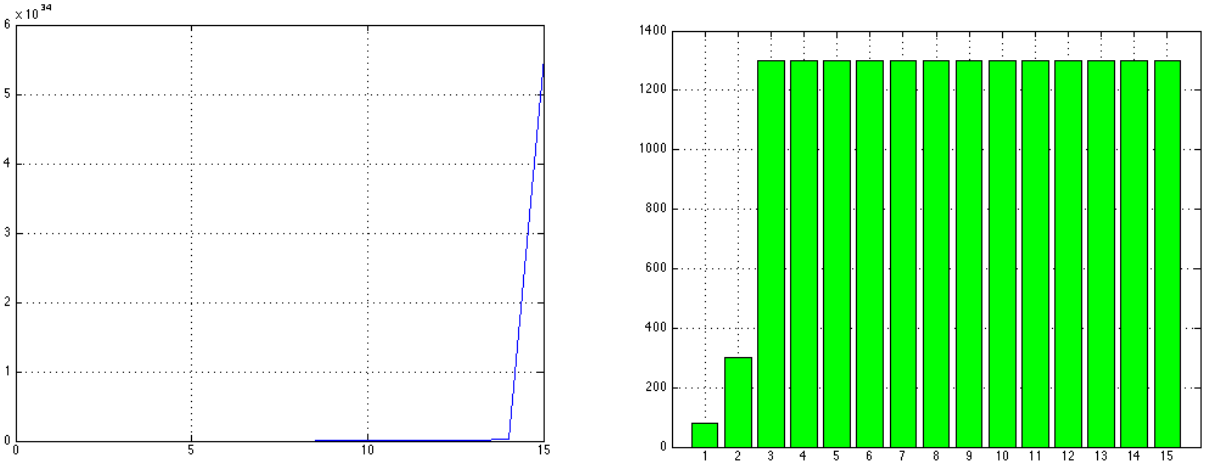

Figura 97: Detalle (1) Resultado Prueba 7.

En este caso, la fuente responde correctamente al patrón de potencias enviado en casi todo momento, a excepción de un error que se es debido al cambio de potencia de 400W a 1300W, sin embargo se vuelve a la normalidad rápidamente tal y como se puede ver en la siguiente captura.

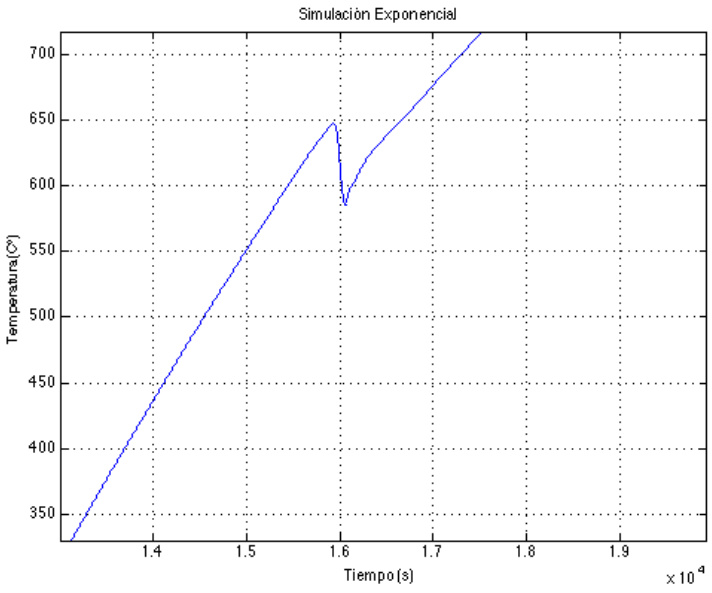

Figura 98: Detalle (2) Resultado Prueba 7.

Terminada la transmisión se alcanza una temperatura máxima de 810ºC que se mantiene durante 1,5 segundos y comienza a descender una vez apagada la fuente y se cierra la comunicación.

5.9 Prueba 8.

La última de las pruebas que se van a exponer se realizó para la comprobación del funcionamiento de la 'Interfaz Puertos'. Los valores introducidos en la interfaz son los siguientes.

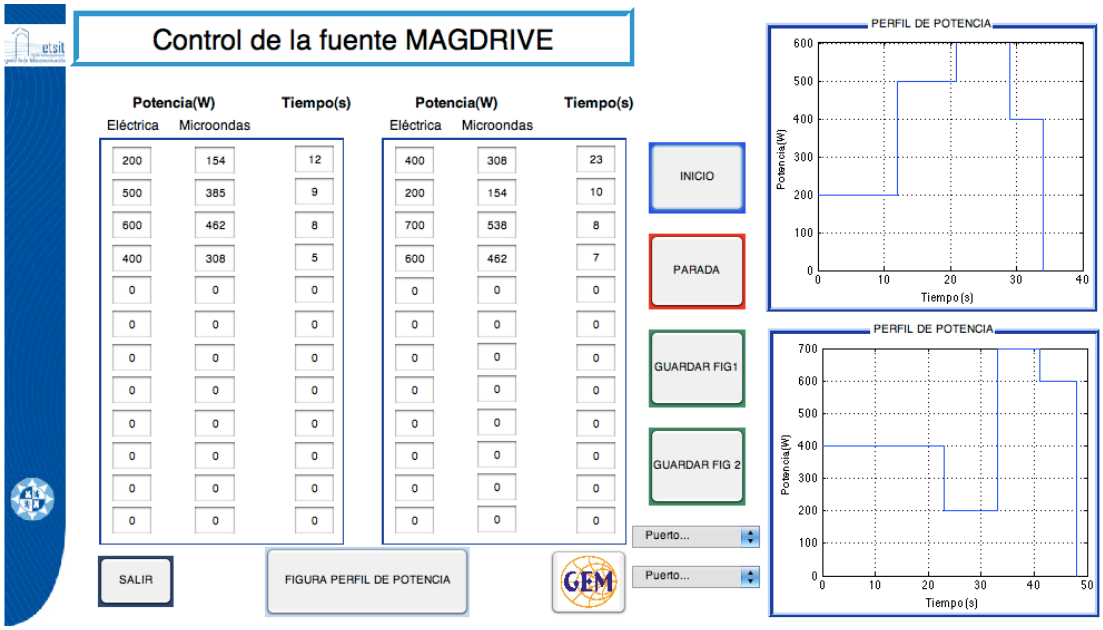

Figura 99: Interfaz Puertos Prueba 8.

Para este modelo no se pudieron tomar medidas en el laboratorio, ya que no se disponía de dos fuentes MAGDRIVE1000, por lo tanto sólo se pudo comprobar su correcto funcionamiento.

Debido a las limitaciones de material para la realización de las pruebas en el laboratorio, el método de medida se ha basado en la realización de diversas pruebas cambiando los puertos de comunicación con la fuente.

En todas las pruebas se pudo ver como sólo se transmitía el primer valor del 'Perfil de potencia 1', con el envío de la respuesta de la fuente al PC indicando la correcta recepción. El siguiente valor que se debía enviar correspondía con el primer valor del 'Perfil de potencia 2' que se envía por el otro puerto de comunicación, esperando una respuesta de la fuente (en este caso inexistente) para proceder al envío del segundo valor de potencia del 'Perfil de potencia 1', y así sucesivamente

Después del primer envío, el sistema se para, ya que para evitar problemas en la transmisión, siempre se deja un tiempo antes de abortar la comunicación tal y como puede comprobarse en el código adjunto en el anexo.

Terminado el tiempo de espera, al no recibir respuesta de la segunda fuente, se envían comandos de apagado por ambos puertos y se cierra la comunicación.

Cada uno de los sucesos descritos anteriormente, eran los que se habían supuesto que en un principio iban a ocurrir. Por lo tanto se ha considerado correcto el funcionamiento de la 'Interfaz puertos'.

6. Conclusiones y líneas futuras.

Durante el desarrollo de este proyecto se han implementado 3 interfaces con el objetivo de controlar de una forma sencilla para el usuario la fuente MAGDRIVE1000, generando distintos patrones de potencia con el fin de calentar objetos.

Los resultados obtenidos en las pruebas realizadas en el laboratorio resultaron satisfactorios, ya que se cumplía el objetivo de poder controlar la fuente. Se debe tener en cuenta, que el control no puede ser absoluto debido a las limitaciones que la fuente presenta por defecto, tal y como se revisaron dentro del apartado correspondiente a la explicación de la fuente Magdrive1000.

Además de las limitaciones que ya se saben que posee la fuente, con las pruebas realizadas se han descubierto otras nuevas, como la limitación de ésta a los cambios de potencia veloces de forma continuada en los niveles de potencia más bajos, tal y como se comprobó al usar la interfaz de funciones.

Por lo tanto se puede decir que tras el detenido análisis de cada una de las interfaces, los resultados obtenidos ha sido satisfactorios y los objetivos planteados al inicio del proyecto se han cumplido.

Las limitaciones que se han detectado en el uso de la fuente podrían ser nuevas líneas de trabajo en las que se podría avanzar, mejorando así su funcionamiento y pudiendo obtener un rendimiento mucho mayor, optimizando los procesos de calentamiento.

7. Bibliografía.

Aplicaciones Industriales del calentamiento asistido por microondas. Juan Monzó Cabrera, Alejandro Díaz Morcillo, Juan Luís Pedreño Molina, José Manuel Catalá Civera, Antonio José Lozano Guerrero, Francisco Javier Clemente Fernández. UPCT.

Principios Fundamentales y Aplicaciones del Calentamiento por Microondas. Alejandro Díaz Morcillo, Juan Monzó Cabrera, Elsa Dominguez Tortajada, MªEugenia Requena Pérez.

Graphics and GUIs with Matlab third edition . Patrick Marchand O.Thomas Holland. Chapman & Hall/CRC.

Dipolar Magdrive1000 1040-042-C2 User Manual.

DipolarMagdrive1000 New Technology for industrial magnetrón Power Supply.

Dipolar RS485 Interface.

1010W CW Magnetron TYPE: 2M244-M23 (Panasonic).

Lectura Guia microondas universidad EATIF abierta al mundo.

IEE standart dictionarian of Electrical on Electronics Terms ANSI std1001947.

http://www.acm-usa.com/Pages/Materials/detail/Materials/0/1

http://www.gallawa.com/microtech/historia-microonda.html

http://www.scribd.com.70539_12b

http://blogs.mathworks.com/videos

8. Anexo.

En este apartado se exponen los códigos de cada una de las interfaces.

8.1 Código Interfaz Simple.

```
function varargout = Interfaz_simple_final(varargin)

%
% Universidad Politécnica De Cartagena (upct)
% Grupo de investigación GEM (Grupo de Electromagnetismo y Materia)
%
% Proyecto Final de Carrera Ing.Superior de Telecomunicación
% Título: Realización De Un Programa Con Interfaz De Usuario Para La
% Generación De Patrones De Potencia Arbitrarios Con La Fuente De Alimentación Magdrive
%
% Alumno: Pablo Bermúdez Alemán
% Director de Proyecto: Juán Monzó Cabrera
% Codirector: Francisco Javier Clemente Fernández
%
% INTERFAZ_SIMPLE_FINAL M-file for Interfaz_simple_final.fig
%      INTERFAZ_SIMPLE_FINAL, by itself, creates a new INTERFAZ_SIMPLE_FINAL or raises
the existing
%      singleton*.
%
%      H = INTERFAZ_SIMPLE_FINAL returns the handle to a new INTERFAZ_SIMPLE_FINAL or
the handle to
%      the existing singleton*.
%
%      INTERFAZ_SIMPLE_FINAL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFAZ_SIMPLE_FINAL.M with the given input arguments.
%
%      INTERFAZ_SIMPLE_FINAL('Property','Value',...) creates a new INTERFAZ_SIMPLE_FINAL
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Interfaz_simple_final_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Interfaz_simple_final_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Interfaz_simple_final

% Last Modified by GUIDE v2.5 27-Dec-2011 18:18:46

% Begin initialization code - DO NOT EDIT
clc

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_simple_final_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_simple_final_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


% End initialization code - DO NOT EDIT
```

```
% --- Executes just before Interfaz_simple_final is made visible.
function Interfaz_simple_final_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interfaz_simple_final (see VARARGIN)

% Choose default command line output for Interfaz_simple_final
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


%Coloca una imagen en cada botÛn
[a,map]=imread('logogem.jpg');
[r,c,d]=size(a);
x=ceil(r/50);
y=ceil(c/60);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.GEM,'CData',g);

puertos=instrhwinfo('serial');
puertos=puertos.AvailableSerialPorts;
set(handles.ports,'String',['Puerto...'; puertos]);

%captura

handles.output = hObject;
% Update handles structure
handles.rgb = [];



% UIWAIT makes Interfaz_simple_final wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_simple_final_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% Condiciones para introducir los datos de Tiempos y Potencias


% ****************************
% ****Condiciones Potencias****
% ****************************


function varargout = p1_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p1,'string'));

if isnan(c1)

    c1=0;

    set(handles.p1,'String', c1);
    set(handles.p_1,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0
```

```
    c1=0;

    set(handles.p1,'String', c1);
    set(handles.p_1,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p1,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p1,'string'));

c1 = round(c1/1.3);

set(handles.p_1,'String', c1);


function varargout = p2_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p2,'string'));
if isnan(c1)

    c1=0;

    set(handles.p2,'String', c1);
    set(handles.p_2,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p2,'String', c1);
    set(handles.p_2,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p2,'string'));
```

```
c1 = round(c1/1.3);

set(handles.p_2,'String', c1);


function varargout = p3_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p3,'string'));
if isnan(c1)

    c1=0;

    set(handles.p3,'String', c1);
    set(handles.p_3,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p3,'String', c1);
    set(handles.p_3,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p3,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p3,'string'));

c1 = round(c1/1.3);

set(handles.p_3,'String', c1);

function varargout = p4_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p4,'string'));
if isnan(c1)

    c1=0;

    set(handles.p4,'String', c1);
    set(handles.p_4,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p4,'String', c1);
    set(handles.p_4,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
```

```
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p4,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p4,'string'));

c1 = round(c1/1.3);

set(handles.p_4,'String', c1);

function varargout = p5_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p5,'string'));
if isnan(c1)

    c1=0;

    set(handles.p5,'String', c1);
    set(handles.p_5,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p5,'String', c1);
    set(handles.p_5,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p5,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p5,'string'));

c1 = round(c1/1.3);

set(handles.p_5,'String', c1);

function varargout = p6_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p6,'string'));
if isnan(c1)
```

```
    c1=0;

    set(handles.p6,'String', c1);
    set(handles.p_6,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p6,'String', c1);
    set(handles.p_6,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p6,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p6,'string'));

c1 = round(c1/1.3);

set(handles.p_6,'String', c1);


function varargout = p7_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p7,'string'));
if isnan(c1)

    c1=0;

    set(handles.p7,'String', c1);
    set(handles.p_7,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p7,'String', c1);
    set(handles.p_7,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
```

```
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p7,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p7,'string'));

c1 = round(c1/1.3);

set(handles.p_7,'String', c1);

function varargout = p8_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p8,'string'));
if isnan(c1)

    c1=0;

    set(handles.p8,'String', c1);
    set(handles.p_8,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p8,'String', c1);
    set(handles.p_8,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p8,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p8,'string'));

c1 = round(c1/1.3);

set(handles.p_8,'String', c1);

function varargout = p9_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p9,'string'));
if isnan(c1)

    c1=0;

    set(handles.p9,'String', c1);
    set(handles.p_9,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p9,'String', c1);
```

```
    set(handles.p_9,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p9,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p9,'string'));

c1 = round(c1/1.3);

set(handles.p_9,'String', c1);

function varargout = p10_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p10,'string'));
if isnan(c1)

    c1=0;

    set(handles.p10,'String', c1);
    set(handles.p10,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p10,'String', c1);
    set(handles.p10,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p10,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p10,'string'));

c1 = round(c1/1.3);

set(handles.p_10,'String', c1);

function varargout = p11_Callback(h, eventdata, handles, varargin)
```

```
c1 = str2double(get(handles.p11,'string'));
if isnan(c1)

    c1=0;

    set(handles.p11,'String', c1);
    set(handles.p11,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p11,'String', c1);
    set(handles.p11,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p11,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p11,'string'));

c1 = round(c1/1.3);

set(handles.p_11,'String', c1);

function varargout = p12_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p12,'string'));
if isnan(c1)

    c1=0;

    set(handles.p12,'String', c1);
    set(handles.p12,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p12,'String', c1);
    set(handles.p12,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
```

```
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p12,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p12,'string'));

c1 = round(c1/1.3);

set(handles.p_12,'String', c1);


% ******************
% Condiciones Tiempos
% ******************


function varargout = t1_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t1,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t2_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t3_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t3,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t4_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t4,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t5_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t5,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t6_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t6,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t7_Callback(h, eventdata, handles, varargin)
```

```
c1 = str2double(get(handles.t7,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t8_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t8,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t9_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t9,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t10_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t10,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t11_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t11,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t12_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t12,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



% --- Executes during object creation, after setting all properties.
function p1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes during object creation, after setting all properties.
function p11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_1_Callback(hObject, eventdata, handles)
% hObject    handle to p_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_1 as text
%        str2double(get(hObject,'String')) returns contents of p_1 as a double



% --- Executes during object creation, after setting all properties.
function p_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_2 as text
%        str2double(get(hObject,'String')) returns contents of p_2 as a double


% --- Executes during object creation, after setting all properties.
function p_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function p_3_Callback(hObject, eventdata, handles)
% hObject    handle to p_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_3 as text
%        str2double(get(hObject,'String')) returns contents of p_3 as a double


% --- Executes during object creation, after setting all properties.
function p_3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_4_Callback(hObject, eventdata, handles)
% hObject    handle to p_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_4 as text
%        str2double(get(hObject,'String')) returns contents of p_4 as a double


% --- Executes during object creation, after setting all properties.
function p_4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_5_Callback(hObject, eventdata, handles)
% hObject    handle to p_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_5 as text
%        str2double(get(hObject,'String')) returns contents of p_5 as a double


% --- Executes during object creation, after setting all properties.
function p_5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_6_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to p_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_6 as text
%        str2double(get(hObject,'String')) returns contents of p_6 as a double


% --- Executes during object creation, after setting all properties.
function p_6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_7_Callback(hObject, eventdata, handles)
% hObject    handle to p_7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_7 as text
%        str2double(get(hObject,'String')) returns contents of p_7 as a double


% --- Executes during object creation, after setting all properties.
function p_7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_8_Callback(hObject, eventdata, handles)
% hObject    handle to p_8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_8 as text
%        str2double(get(hObject,'String')) returns contents of p_8 as a double


% --- Executes during object creation, after setting all properties.
function p_8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_9_Callback(hObject, eventdata, handles)
% hObject    handle to p_9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_9 as text
```

```
%        str2double(get(hObject,'String')) returns contents of p_9 as a double


% --- Executes during object creation, after setting all properties.
function p_9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_10_Callback(hObject, eventdata, handles)
% hObject    handle to p_10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_10 as text
%        str2double(get(hObject,'String')) returns contents of p_10 as a double


% --- Executes during object creation, after setting all properties.
function p_10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_11_Callback(hObject, eventdata, handles)
% hObject    handle to p_11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_11 as text
%        str2double(get(hObject,'String')) returns contents of p_11 as a double


% --- Executes during object creation, after setting all properties.
function p_11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_12_Callback(hObject, eventdata, handles)
% hObject    handle to p_12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_12 as text
%        str2double(get(hObject,'String')) returns contents of p_12 as a double


% --- Executes during object creation, after setting all properties.
function p_12_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to p_12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% ******************
% ***** INICIO ******
% ******************


% --- Executes on button press in Inicio.
function Inicio_Callback(hObject, eventdata, handles)
% hObject    handle to Inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Inicio
global pot1 tiempo1 pot2 tiempo2 pot3 tiempo3 pot4 tiempo4 pot5 tiempo5 pot6 tiempo6
pot7 tiempo7 pot8 tiempo8
global pot9 tiempo9 pot10 tiempo10 pot11 tiempo11 pot12 tiempo12

pot1=str2double(get(handles.p1,'String'));
tiempo1=str2double(get(handles.t1,'String'));

pot2=str2double(get(handles.p2,'String'));
tiempo2=str2double(get(handles.t2,'String'));

pot3=str2double(get(handles.p3,'String'));
tiempo3=str2double(get(handles.t3,'String'));

pot4=str2double(get(handles.p4,'String'));
tiempo4=str2double(get(handles.t4,'String'));

pot5=str2double(get(handles.p5,'String'));
tiempo5=str2double(get(handles.t5,'String'));

pot6=str2double(get(handles.p6,'String'));
tiempo6=str2double(get(handles.t6,'String'));

pot7=str2double(get(handles.p7,'String'));
tiempo7=str2double(get(handles.t7,'String'));

pot8=str2double(get(handles.p8,'String'));
tiempo8=str2double(get(handles.t8,'String'));

pot9=str2double(get(handles.p9,'String'));
tiempo9=str2double(get(handles.t9,'String'));

pot10=str2double(get(handles.p10,'String'));
tiempo10=str2double(get(handles.t10,'String'));

pot11=str2double(get(handles.p11,'String'));
tiempo11=str2double(get(handles.t11,'String'));

pot12=str2double(get(handles.p12,'String'));
tiempo12=str2double(get(handles.t12,'String'));

potencias=[pot1 pot2 pot3 pot4 pot5 pot6 pot7 pot8 pot9 pot10 pot11 pot12];
tiempos=[tiempo1 tiempo2 tiempo3 tiempo4 tiempo5 tiempo6 tiempo7 tiempo8 tiempo9
tiempo10 tiempo11 tiempo12];

set(handles.perfil,'HandleVisibility','ON')
axes(handles.perfil)


a = tiempo1+tiempo2;
b = tiempo1+tiempo2+tiempo3;
c = tiempo1+tiempo2+tiempo3+tiempo4;
d = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5;
```

```
e = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6;
f = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7;
g = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8;
h = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9;
i = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10;
j =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1;
k =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1+tiempo12;



eje_tiempo=[linspace(0,tiempo1,100) linspace(tiempo1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];


eje_pot=[pot1*ones(1,100) pot2*ones(1,100) pot3*ones(1,100) pot4*ones(1,100)
pot5*ones(1,100) pot6*ones(1,100) pot7*ones(1,100) pot8*ones(1,100) pot9*ones(1,100)
pot10*ones(1,100) pot11*ones(1,100) pot12*ones(1,100)];


plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)');grid

set(handles.perfil,'HandleVisibility','OFF')


[valores,indices]=find(potencias);
global serie ans2
% Se establece la condiciÛn de parada NO
ans2 = 'No';

fopen(serie)
set(serie,'RequestToSend','on');
set(serie,'DataTerminalReady','off');
set(serie,'RequestToSend','off');
set(serie,'RequestToSend','on');
set(serie,'BaudRate',2400);
set(serie,'Terminator','CR')

for i=1:max(indices)
    str1=[128 37 potencias(i)/10];
    chk=mod(sum(str1),256);
    str1=[str1 chk];
    for j=1:tiempos(i)

        % Se comprueba en cada iteraciÛn la condiciÛn de parada
        if strcmp(ans2,'Si')
            return;
        end
        tic
        fwrite(serie,str1);
        fread(serie,3)
        fwrite(serie,[64 24 0 88])

        % Se comprueba el estado de alarma de la fuente
        alarma=fread(serie,3);
        alarma=alarma(2);
        alarma=dec2bin(alarma);
        ind=find(alarma=='0');
        alarma=max(ind)-1;
        switch alarma
            case '0'
                warndlg('Fallo,  control del Filamento','GEM');
            case '1'
                warndlg('Fallo,  control de Potencia AnÛdica','GEM');
            case '2'
                warndlg('Fallo, control de Temperatura','GEM');
            case '3'
                warndlg('N.C','GEM');
            case '4'
                warndlg('RegulaciÛn de Temperatura en progreso','GEM');
            case '5'
                warndlg('Fallo, control PFC','GEM');
```

```
            case '6'
                warndlg('N.C','GEM');
            case '7'
                warndlg('Unidad deshabilitada debido a una alarma','GEM');

        end
        pause(0.94)
        toc
    end
end

% Se envÌa  SIEMPRE un comando de apagado de la fuente despuÈs de la transmisiÛn
str1=[128 37 0];
chk=sum(str1);
str1=[str1 chk];
fwrite(serie,str1);
fread(serie,3)
pause(0.94)

fclose(serie)


% ******************
% ***** PARADA ******
% ******************

% --- Executes on button press in Parada.
function Parada_Callback(hObject, eventdata, handles)
% hObject    handle to Parada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Parada

global ans2

ans2=questdlg('øDesea detener la transmisiÛn?','GEM','Si','No','No');

if strcmp(ans2,'No')
return;
end

if strcmp(ans2,'Si')

global serie

    % Se envÌa el comando de parada
    str1=[128 37 0];
    chk=sum(str1);
    str1=[str1 chk];
        tic
        fwrite(serie,str1);
        fread(serie,3)
        pause(0.94)
        toc
fclose(serie)

return;
end

clearall,clc


% ******************************
% ***** PERFIL DE POTENCIA ******
% ******************************


% --- Executes on button press in Figura_Perfil_Potencia.
function Figura_Perfil_Potencia_Callback(hObject, eventdata, handles)
% hObject    handle to Figura_Perfil_Potencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Figura_Perfil_Potencia
global pot1 tiempo1 pot2 tiempo2 pot3 tiempo3 pot4 tiempo4 pot5 tiempo5 pot6 tiempo6
pot7 tiempo7 pot8 tiempo8 pot9 tiempo9 pot10 tiempo10 pot11 tiempo11 pot12 tiempo12
```

```
pot1=str2double(get(handles.p1,'String'));
tiempo1=str2double(get(handles.t1,'String'));

pot2=str2double(get(handles.p2,'String'));
tiempo2=str2double(get(handles.t2,'String'));

pot3=str2double(get(handles.p3,'String'));
tiempo3=str2double(get(handles.t3,'String'));

pot4=str2double(get(handles.p4,'String'));
tiempo4=str2double(get(handles.t4,'String'));

pot5=str2double(get(handles.p5,'String'));
tiempo5=str2double(get(handles.t5,'String'));

pot6=str2double(get(handles.p6,'String'));
tiempo6=str2double(get(handles.t6,'String'));

pot7=str2double(get(handles.p7,'String'));
tiempo7=str2double(get(handles.t7,'String'));

pot8=str2double(get(handles.p8,'String'));
tiempo8=str2double(get(handles.t8,'String'));

pot9=str2double(get(handles.p9,'String'));
tiempo9=str2double(get(handles.t9,'String'));

pot10=str2double(get(handles.p10,'String'));
tiempo10=str2double(get(handles.t10,'String'));

pot11=str2double(get(handles.p11,'String'));
tiempo11=str2double(get(handles.t11,'String'));

pot12=str2double(get(handles.p12,'String'));
tiempo12=str2double(get(handles.t12,'String'));

set(handles.perfil,'HandleVisibility','ON')
axes(handles.perfil)


a = tiempo1+tiempo2;
b = tiempo1+tiempo2+tiempo3;
c = tiempo1+tiempo2+tiempo3+tiempo4;
d = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5;
e = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6;
f = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7;
g = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8;
h = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9;
i = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10;
j =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1;
k =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1+tiempo12;



eje_tiempo=[linspace(0,tiempo1,100) linspace(tiempo1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];


eje_pot=[pot1*ones(1,100) pot2*ones(1,100) pot3*ones(1,100) pot4*ones(1,100)
pot5*ones(1,100) pot6*ones(1,100) pot7*ones(1,100) pot8*ones(1,100) pot9*ones(1,100)
pot10*ones(1,100) pot11*ones(1,100) pot12*ones(1,100)];


plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)');grid

set(handles.perfil,'HandleVisibility','OFF')
```

```
function edit26_Callback(hObject, eventdata, handles)
% hObject    handle to Temp_result (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Temp_result as text
%        str2double(get(hObject,'String')) returns contents of Temp_result as a double


% --- Executes on button press in togglebutton5.
function togglebutton5_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton5


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% Se introduce el fondo del programa

% --- Executes during object creation, after setting all properties.
function upct_CreateFcn(hObject, eventdata, handles)
% hObject    handle to upct (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
axes(hObject)
imshow('Fondo.jpg')
% Hint: place code in OpeningFcn to populate upct




% ******************
% ***** Salida *****
% ******************


% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

ans=questdlg('øDesea salir del programa?','GEM','Si','No','No');
if strcmp(ans,'No')
return;
end
clear,clc,close all



% --- Executes on button press in GEM.
function GEM_Callback(hObject, eventdata, handles)
% hObject    handle to GEM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

msgbox('Grupo de Electromagnetisto y materia (UPCT)',' GEM ');


% --- Executes on button press in guardar.
function guardar_Callback(hObject, eventdata, handles)
% hObject    handle to guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% --- FUNCI”N DEL BOT”N "GUARDAR"

% formatos = {'*.jpg','JPEG (*.jpg)';'*.tif','TIFF (*.tif)'};
% [nomb,ruta] = uiputfile(formatos,'GUARDAR HISTOGRAMA');
% if nomb==0, return, end
% % % Crear nueva figura
%
% figura = figure;
% %Unidades y posiciÛn
% unidades  = get(handles.perfil,'Units');
% posicion   = get(handles.perfil,'Position');
% objeto_2  = copyobj(handles.perfil,figura);
%
% a=get(handles.perfil)
% %Ajustar la nueva figura
%
% set(figura,'Units',unidades);
% set(figura,'Position',[15 5 posicion(1)+60 posicion(2)+24]);
%
% % Guardar la gr·fica
% saveas(figura,[ruta nomb])
% % Cerrar figura
% close(figura)

% Crear nueva figura
figura = figure;

% Unidades y posiciÛn
unidades  = get(handles.perfil,'Units');
posicion   = get(handles.perfil,'Position');
fig  = copyobj(handles.perfil,figura);

set(figura,'Units',unidades);
set(figura,'Position',[10 5 posicion(3)+12 posicion(4)+5])


function ports_Callback(hObject, eventdata, handles)
% hObject    handle to ports (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns ports contents as cell array
%        contents{get(hObject,'Value')} returns selected item from ports

global serie
puertos=instrhwinfo('serial');
puertos=puertos.AvailableSerialPorts;
puerto_serie=get(handles.ports, 'Value');
% Crea puerto serie para comunicacion
serie=serial(puertos{puerto_serie-1});
set(serie,'RequestToSend','on');
set(serie,'DataTerminalReady','off');
set(serie,'BaudRate',2400);
set(serie,'Terminator','CR')



% --- Executes during object creation, after setting all properties.
function perfil_CreateFcn(hObject, eventdata, handles)
% hObject    handle to perfil (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate perfil


% --- Executes during object creation, after setting all properties.
function Inicio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function ports_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ports (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

8.2 Código Interfaz Puertos.

```
function varargout = Interfaz_simple_final_puertos(varargin)

%
% Universidad PolitÈcnica De Cartagena (upct)
% Grupo de investigaciÛn GEM (Grupo de Electromagnetismo y Materia)
%
% Proyecto Final de Carrera Ing.Superior de TelecomunicaciÛn
% TÌtulo: RealizaciÛn De Un Programa Con Interfaz De Usuario Para La
% GeneraciÛn De Patrones De Potencia Arbitratios Con La Fuente De AlimentaciÛn Magdrive
%
% Alumno: Pablo Berm·dez Alem·n
% Director de Proyecto: Ju·n MonzÛ Cabrera
% Codirector: Francisco Javier Clemente Fern·ndez
%
% INTERFAZ_SIMPLE_FINAL_PUERTOS M-file for Interfaz_simple_final_puertos.fig
%      INTERFAZ_SIMPLE_FINAL_PUERTOS, by itself, creates a new
INTERFAZ_SIMPLE_FINAL_PUERTOS or raises the existing
%      singleton*.
%
%      H = INTERFAZ_SIMPLE_FINAL_PUERTOS returns the handle to a new
INTERFAZ_SIMPLE_FINAL_PUERTOS or the handle to
%      the existing singleton*.
%
%      INTERFAZ_SIMPLE_FINAL_PUERTOS('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in INTERFAZ_SIMPLE_FINAL_PUERTOS.M with the given input
arguments.
%
%      INTERFAZ_SIMPLE_FINAL_PUERTOS('Property','Value',...) creates a new
INTERFAZ_SIMPLE_FINAL_PUERTOS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Interfaz_simple_final_puertos_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Interfaz_simple_final_puertos_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Interfaz_simple_final_puertos

% Last Modified by GUIDE v2.5 15-Dec-2011 16:46:09

% Begin initialization code - DO NOT EDIT
clc

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_simple_final_puertos_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_simple_final_puertos_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


% End initialization code - DO NOT EDIT


% --- Executes just before Interfaz_simple_final_puertos is made visible.
function Interfaz_simple_final_puertos_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interfaz_simple_final_puertos (see VARARGIN)

% Choose default command line output for Interfaz_simple_final_puertos
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


%Coloca una imagen en cada botÛn
[a,map]=imread('logogem.jpg');
[r,c,d]=size(a);
x=ceil(r/50);
y=ceil(c/60);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.GEM,'CData',g);


puertos=instrhwinfo('serial');
puertos=puertos.AvailableSerialPorts;
set(handles.ports,'String',['Puerto...'; puertos]);


set(handles.ports_1,'String',['Puerto...'; puertos]);




% UIWAIT makes Interfaz_simple_final_puertos wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_simple_final_puertos_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% Condiciones para introducir los datos de Tiempos y Potencias


% ****************************
% ****Condiciones Potencias****
% ****************************


function varargout = p1_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p1,'string'));

if isnan(c1)

    c1=0;

    set(handles.p1,'String', c1);
    set(handles.p_1,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;
```

```
    set(handles.p1,'String', c1);
    set(handles.p_1,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p1,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p1,'string'));

c1 = round(c1/1.3);

set(handles.p_1,'String', c1);


function varargout = p2_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p2,'string'));
if isnan(c1)

    c1=0;

    set(handles.p2,'String', c1);
    set(handles.p_2,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p2,'String', c1);
    set(handles.p_2,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p2,'string'));

c1 = round(c1/1.3);
```

```
set(handles.p_2,'String', c1);


function varargout = p3_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p3,'string'));
if isnan(c1)

    c1=0;

    set(handles.p3,'String', c1);
    set(handles.p_3,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p3,'String', c1);
    set(handles.p_3,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p3,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p3,'string'));

c1 = round(c1/1.3);

set(handles.p_3,'String', c1);

function varargout = p4_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p4,'string'));
if isnan(c1)

    c1=0;

    set(handles.p4,'String', c1);
    set(handles.p_4,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p4,'String', c1);
    set(handles.p_4,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p4,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p4,'string'));

c1 = round(c1/1.3);

set(handles.p_4,'String', c1);

function varargout = p5_Callback(h, eventdata, handles, varargin)


c1 = str2double(get(handles.p5,'string'));
if isnan(c1)

    c1=0;

    set(handles.p5,'String', c1);
    set(handles.p_5,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p5,'String', c1);
    set(handles.p_5,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p5,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p5,'string'));

c1 = round(c1/1.3);

set(handles.p_5,'String', c1);

function varargout = p6_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p6,'string'));
if isnan(c1)

    c1=0;
```

```
    set(handles.p6,'String', c1);
    set(handles.p_6,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p6,'String', c1);
    set(handles.p_6,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p6,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p6,'string'));

c1 = round(c1/1.3);

set(handles.p_6,'String', c1);


function varargout = p7_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p7,'string'));
if isnan(c1)

    c1=0;

    set(handles.p7,'String', c1);
    set(handles.p_7,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p7,'String', c1);
    set(handles.p_7,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end
```

```
set(handles.p7,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p7,'string'));

c1 = round(c1/1.3);

set(handles.p_7,'String', c1);

function varargout = p8_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p8,'string'));
if isnan(c1)

    c1=0;

    set(handles.p8,'String', c1);
    set(handles.p_8,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p8,'String', c1);
    set(handles.p_8,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p8,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p8,'string'));

c1 = round(c1/1.3);

set(handles.p_8,'String', c1);

function varargout = p9_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p9,'string'));
if isnan(c1)

    c1=0;

    set(handles.p9,'String', c1);
    set(handles.p_9,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p9,'String', c1);
    set(handles.p_9,'String', c1);
```

```
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p9,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p9,'string'));

c1 = round(c1/1.3);

set(handles.p_9,'String', c1);

function varargout = p10_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p10,'string'));
if isnan(c1)

    c1=0;

    set(handles.p10,'String', c1);
    set(handles.p10,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p10,'String', c1);
    set(handles.p10,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p10,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p10,'string'));

c1 = round(c1/1.3);

set(handles.p_10,'String', c1);

function varargout = p11_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p11,'string'));
```

```
if isnan(c1)

    c1=0;

    set(handles.p11,'String', c1);
    set(handles.p11,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p11,'String', c1);
    set(handles.p11,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p11,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p11,'string'));

c1 = round(c1/1.3);

set(handles.p_11,'String', c1);

function varargout = p12_Callback(h, eventdata, handles, varargin)

c1 = str2double(get(handles.p12,'string'));
if isnan(c1)

    c1=0;

    set(handles.p12,'String', c1);
    set(handles.p12,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p12,'String', c1);
    set(handles.p12,'String', c1);


errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end
```

```
if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p12,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p12,'string'));

c1 = round(c1/1.3);

set(handles.p_12,'String', c1);


% ********************
% Condiciones Tiempos
% ********************


function varargout = t1_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t1,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t2_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t3_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t3,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t4_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t4,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t5_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t5,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t6_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t6,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t7_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t7,'string'));
if isnan(c1)
```

```
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t8_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t8,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t9_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t9,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t10_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t10,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t11_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t11,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t12_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t12,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



% --- Executes during object creation, after setting all properties.
function p1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function p8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes during object creation, after setting all properties.
function p11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
```

```
function t11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_1_Callback(hObject, eventdata, handles)
% hObject    handle to p_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_1 as text
%        str2double(get(hObject,'String')) returns contents of p_1 as a double



% --- Executes during object creation, after setting all properties.
function p_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_2 as text
%        str2double(get(hObject,'String')) returns contents of p_2 as a double


% --- Executes during object creation, after setting all properties.
function p_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function p_3_Callback(hObject, eventdata, handles)
% hObject    handle to p_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_3 as text
%        str2double(get(hObject,'String')) returns contents of p_3 as a double


% --- Executes during object creation, after setting all properties.
function p_3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_4_Callback(hObject, eventdata, handles)
% hObject    handle to p_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_4 as text
%        str2double(get(hObject,'String')) returns contents of p_4 as a double


% --- Executes during object creation, after setting all properties.
function p_4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_5_Callback(hObject, eventdata, handles)
% hObject    handle to p_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_5 as text
%        str2double(get(hObject,'String')) returns contents of p_5 as a double


% --- Executes during object creation, after setting all properties.
function p_5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_6_Callback(hObject, eventdata, handles)
% hObject    handle to p_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_6 as text
%        str2double(get(hObject,'String')) returns contents of p_6 as a double


% --- Executes during object creation, after setting all properties.
function p_6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_7_Callback(hObject, eventdata, handles)
% hObject    handle to p_7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_7 as text
%        str2double(get(hObject,'String')) returns contents of p_7 as a double


% --- Executes during object creation, after setting all properties.
function p_7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_8_Callback(hObject, eventdata, handles)
% hObject    handle to p_8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_8 as text
%        str2double(get(hObject,'String')) returns contents of p_8 as a double


% --- Executes during object creation, after setting all properties.
function p_8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_9_Callback(hObject, eventdata, handles)
% hObject    handle to p_9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_9 as text
%        str2double(get(hObject,'String')) returns contents of p_9 as a double
```

```
% --- Executes during object creation, after setting all properties.
function p_9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_10_Callback(hObject, eventdata, handles)
% hObject    handle to p_10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_10 as text
%        str2double(get(hObject,'String')) returns contents of p_10 as a double


% --- Executes during object creation, after setting all properties.
function p_10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_11_Callback(hObject, eventdata, handles)
% hObject    handle to p_11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_11 as text
%        str2double(get(hObject,'String')) returns contents of p_11 as a double


% --- Executes during object creation, after setting all properties.
function p_11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_12_Callback(hObject, eventdata, handles)
% hObject    handle to p_12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_12 as text
%        str2double(get(hObject,'String')) returns contents of p_12 as a double


% --- Executes during object creation, after setting all properties.
function p_12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% Condiciones para introducir los datos de Tiempos y Potencias


% ****************************
% ****Condiciones Potencias**** PARA EL PUERTO 2
% ****************************



function p1_2_Callback(hObject, eventdata, handles)
% hObject    handle to p1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p1_2 as text
%        str2double(get(hObject,'String')) returns contents of p1_2 as a double

c1 = str2double(get(handles.p1_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p1_2,'String', c1);
    set(handles.p_1_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p1_2,'String', c1);
    set(handles.p_1_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p1_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p1_2,'string'));

c1 = round(c1/1.3);

set(handles.p_1_2,'String', c1);
```

```
function p2_2_Callback(hObject, eventdata, handles)
% hObject    handle to p2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p2_2 as text
%        str2double(get(hObject,'String')) returns contents of p2_2 as a double
c1 = str2double(get(handles.p2_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p2_2,'String', c1);
    set(handles.p_2_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p2_2,'String', c1);
    set(handles.p_2_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p2_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p2_2,'string'));

c1 = round(c1/1.3);

set(handles.p_2_2,'String', c1);



function p3_2_Callback(hObject, eventdata, handles)
% hObject    handle to p3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p3_2 as text
%        str2double(get(hObject,'String')) returns contents of p3_2 as a double

c1 = str2double(get(handles.p3_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p3_2,'String', c1);
    set(handles.p_3_2,'String', c1);
```

```
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p3_2,'String', c1);
    set(handles.p_3_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p3_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p3_2,'string'));

c1 = round(c1/1.3);

set(handles.p_3_2,'String', c1);



function p4_2_Callback(hObject, eventdata, handles)
% hObject    handle to p4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p4_2 as text
%        str2double(get(hObject,'String')) returns contents of p4_2 as a double

c1 = str2double(get(handles.p4_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p4_2,'String', c1);
    set(handles.p_4_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p4_2,'String', c1);
    set(handles.p_4_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end
```

```
if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p4_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p4_2,'string'));

c1 = round(c1/1.3);

set(handles.p_4_2,'String', c1);




function p5_2_Callback(hObject, eventdata, handles)
% hObject    handle to p5_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p5_2 as text
%        str2double(get(hObject,'String')) returns contents of p5_2 as a double

c1 = str2double(get(handles.p5_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p5_2,'String', c1);
    set(handles.p_5_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p5_2,'String', c1);
    set(handles.p_5_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p4_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p5_2,'string'));

c1 = round(c1/1.3);

set(handles.p_5_2,'String', c1);



function p6_2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to p6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p6_2 as text
%        str2double(get(hObject,'String')) returns contents of p6_2 as a double
c1 = str2double(get(handles.p6_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p6_2,'String', c1);
    set(handles.p_6_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p6_2,'String', c1);
    set(handles.p_6_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p6_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p6_2,'string'));

c1 = round(c1/1.3);

set(handles.p_6_2,'String', c1);



function p7_2_Callback(hObject, eventdata, handles)
% hObject    handle to p7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p7_2 as text
%        str2double(get(hObject,'String')) returns contents of p7_2 as a double
c1 = str2double(get(handles.p7_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p7_2,'String', c1);
    set(handles.p_7_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0
```

```
    c1=0;

    set(handles.p7_2,'String', c1);
    set(handles.p_7_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p7_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p7_2,'string'));

c1 = round(c1/1.3);

set(handles.p_7_2,'String', c1);



function p8_2_Callback(hObject, eventdata, handles)
% hObject    handle to p8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p8_2 as text
%        str2double(get(hObject,'String')) returns contents of p8_2 as a double
c1 = str2double(get(handles.p8_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p8_2,'String', c1);
    set(handles.p_8_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p8_2,'String', c1);
    set(handles.p_8_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end
```

```
set(handles.p8_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p8_2,'string'));

c1 = round(c1/1.3);

set(handles.p_8_2,'String', c1);



function p9_2_Callback(hObject, eventdata, handles)
% hObject    handle to p9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p9_2 as text
%        str2double(get(hObject,'String')) returns contents of p9_2 as a double

c1 = str2double(get(handles.p9_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p9_2,'String', c1);
    set(handles.p_9_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p9_2,'String', c1);
    set(handles.p_9_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p9_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p9_2,'string'));

c1 = round(c1/1.3);

set(handles.p_9_2,'String', c1);


function p10_2_Callback(hObject, eventdata, handles)
% hObject    handle to p10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p10_2 as text
%        str2double(get(hObject,'String')) returns contents of p10_2 as a double
c1 = str2double(get(handles.p10_2,'string'));
```

```
if isnan(c1)

    c1=0;

    set(handles.p10_2,'String', c1);
    set(handles.p_10_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p10_2,'String', c1);
    set(handles.p_10_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p10_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p10_2,'string'));

c1 = round(c1/1.3);

set(handles.p_10_2,'String', c1);



function p11_2_Callback(hObject, eventdata, handles)
% hObject    handle to p11_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p11_2 as text
%        str2double(get(hObject,'String')) returns contents of p11_2 as a double

c1 = str2double(get(handles.p11_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p11_2,'String', c1);
    set(handles.p_11_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p11_2,'String', c1);
    set(handles.p_11_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p11_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p11_2,'string'));

c1 = round(c1/1.3);

set(handles.p_11_2,'String', c1);


function p12_2_Callback(hObject, eventdata, handles)
% hObject    handle to p12_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p12_2 as text
%        str2double(get(hObject,'String')) returns contents of p12_2 as a double
c1 = str2double(get(handles.p12_2,'string'));

if isnan(c1)

    c1=0;

    set(handles.p12_2,'String', c1);
    set(handles.p_12_2,'String', c1);


errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end

if c1<0

    c1=0;

    set(handles.p12_2,'String', c1);
    set(handles.p_12_2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

c1=c1/10;
c1=round(c1);
c1=c1*10;

if c1 > 1300
    c1=1300;
    errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input', 'modal')
end

if c1<80 && c1 ~= 0
    c1=80;
    errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input', 'modal')
end

set(handles.p12_2,'String', c1);

%Se establece la potencia en microondas
c1 = str2double(get(handles.p12_2,'string'));

c1 = round(c1/1.3);

set(handles.p_12_2,'String', c1);
```

```
% ******************
% Condiciones Tiempos
% ******************


function t1_2_Callback(hObject, eventdata, handles)
% hObject    handle to t1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t1_2 as text
%        str2double(get(hObject,'String')) returns contents of t1_2 as a double

c1 = str2double(get(handles.t1_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



function t2_2_Callback(hObject, eventdata, handles)
% hObject    handle to t2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t2_2 as text
%        str2double(get(hObject,'String')) returns contents of t2_2 as a double

c1 = str2double(get(handles.t2_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



function t3_2_Callback(hObject, eventdata, handles)
% hObject    handle to t3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t3_2 as text
%        str2double(get(hObject,'String')) returns contents of t3_2 as a double

c1 = str2double(get(handles.t3_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



function t4_2_Callback(hObject, eventdata, handles)
% hObject    handle to t4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t4_2 as text
%        str2double(get(hObject,'String')) returns contents of t4_2 as a double

c1 = str2double(get(handles.t4_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
function t5_2_Callback(hObject, eventdata, handles)
% hObject    handle to t5_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t5_2 as text
%        str2double(get(hObject,'String')) returns contents of t5_2 as a double

c1 = str2double(get(handles.t5_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



function t6_2_Callback(hObject, eventdata, handles)
% hObject    handle to t6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t6_2 as text
%        str2double(get(hObject,'String')) returns contents of t6_2 as a double

c1 = str2double(get(handles.t6_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



function t7_2_Callback(hObject, eventdata, handles)
% hObject    handle to t7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t7_2 as text
%        str2double(get(hObject,'String')) returns contents of t7_2 as a double

c1 = str2double(get(handles.t7_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



function t8_2_Callback(hObject, eventdata, handles)
% hObject    handle to t8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t8_2 as text
%        str2double(get(hObject,'String')) returns contents of t8_2 as a double

c1 = str2double(get(handles.t8_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
function t9_2_Callback(hObject, eventdata, handles)
% hObject    handle to t9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t9_2 as text
%        str2double(get(hObject,'String')) returns contents of t9_2 as a double

c1 = str2double(get(handles.t9_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
function t10_2_Callback(hObject, eventdata, handles)
% hObject    handle to t10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t10_2 as text
%        str2double(get(hObject,'String')) returns contents of t10_2 as a double

c1 = str2double(get(handles.t10_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
function t11_2_Callback(hObject, eventdata, handles)
% hObject    handle to t11_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t11_2 as text
%        str2double(get(hObject,'String')) returns contents of t11_2 as a double

c1 = str2double(get(handles.t11_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
function t12_2_Callback(hObject, eventdata, handles)
% hObject    handle to t12_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of t12_2 as text
%        str2double(get(hObject,'String')) returns contents of t12_2 as a double

c1 = str2double(get(handles.t12_2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
%****************************
%*********Funciones***********
%****************************


% --- Executes during object creation, after setting all properties.
function p1_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p2_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p3_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p4_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p5_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p5_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function p6_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p7_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function p8_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p9_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p10_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p11_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p11_2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p12_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p12_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



%*****************


% --- Executes during object creation, after setting all properties.
function t1_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t2_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t3_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t4_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t5_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t5_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t6_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t7_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t8_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t9_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
```

```
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function t10_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t11_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t11_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t12_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t12_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function p_1_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_1_2 as text
%        str2double(get(hObject,'String')) returns contents of p_1_2 as a double


% --- Executes during object creation, after setting all properties.
function p_1_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% **********************

function p_2_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_2_2 as text
%        str2double(get(hObject,'String')) returns contents of p_2_2 as a double
```

```
% --- Executes during object creation, after setting all properties.
function p_2_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_3_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_3_2 as text
%        str2double(get(hObject,'String')) returns contents of p_3_2 as a double


% --- Executes during object creation, after setting all properties.
function p_3_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_4_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_4_2 as text
%        str2double(get(hObject,'String')) returns contents of p_4_2 as a double


% --- Executes during object creation, after setting all properties.
function p_4_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_4_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_5_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_5_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_5_2 as text
%        str2double(get(hObject,'String')) returns contents of p_5_2 as a double


% --- Executes during object creation, after setting all properties.
function p_5_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_5_2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_6_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_6_2 as text
%        str2double(get(hObject,'String')) returns contents of p_6_2 as a double


% --- Executes during object creation, after setting all properties.
function p_6_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_6_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_7_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_7_2 as text
%        str2double(get(hObject,'String')) returns contents of p_7_2 as a double


% --- Executes during object creation, after setting all properties.
function p_7_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_7_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_8_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_8_2 as text
%        str2double(get(hObject,'String')) returns contents of p_8_2 as a double


% --- Executes during object creation, after setting all properties.
function p_8_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_8_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_9_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_9_2 as text
%        str2double(get(hObject,'String')) returns contents of p_9_2 as a double


% --- Executes during object creation, after setting all properties.
function p_9_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_9_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_10_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_10_2 as text
%        str2double(get(hObject,'String')) returns contents of p_10_2 as a double


% --- Executes during object creation, after setting all properties.
function p_10_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_10_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p_11_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_11_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_11_2 as text
%        str2double(get(hObject,'String')) returns contents of p_11_2 as a double


% --- Executes during object creation, after setting all properties.
function p_11_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_11_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function p_12_2_Callback(hObject, eventdata, handles)
% hObject    handle to p_12_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p_12_2 as text
%        str2double(get(hObject,'String')) returns contents of p_12_2 as a double


% --- Executes during object creation, after setting all properties.
function p_12_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_12_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% ******************
% ***** INICIO ******
% ******************


% --- Executes on button press in Inicio.
function Inicio_Callback(hObject, eventdata, handles)
% hObject    handle to Inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Inicio
global pot1 tiempo1 pot2 tiempo2 pot3 tiempo3 pot4 tiempo4 pot5 tiempo5 pot6 tiempo6
pot7 tiempo7 pot8 tiempo8
global pot9 tiempo9 pot10 tiempo10 pot11 tiempo11 pot12 tiempo12

pot1=str2double(get(handles.p1,'String'));
tiempo1=str2double(get(handles.t1,'String'));

pot2=str2double(get(handles.p2,'String'));
tiempo2=str2double(get(handles.t2,'String'));

pot3=str2double(get(handles.p3,'String'));
tiempo3=str2double(get(handles.t3,'String'));

pot4=str2double(get(handles.p4,'String'));
tiempo4=str2double(get(handles.t4,'String'));

pot5=str2double(get(handles.p5,'String'));
tiempo5=str2double(get(handles.t5,'String'));

pot6=str2double(get(handles.p6,'String'));
tiempo6=str2double(get(handles.t6,'String'));

pot7=str2double(get(handles.p7,'String'));
tiempo7=str2double(get(handles.t7,'String'));

pot8=str2double(get(handles.p8,'String'));
tiempo8=str2double(get(handles.t8,'String'));

pot9=str2double(get(handles.p9,'String'));
tiempo9=str2double(get(handles.t9,'String'));

pot10=str2double(get(handles.p10,'String'));
tiempo10=str2double(get(handles.t10,'String'));

pot11=str2double(get(handles.p11,'String'));
tiempo11=str2double(get(handles.t11,'String'));

pot12=str2double(get(handles.p12,'String'));
tiempo12=str2double(get(handles.t12,'String'));
```

```
potencias=[pot1 pot2 pot3 pot4 pot5 pot6 pot7 pot8 pot9 pot10 pot11 pot12];
tiempos=[tiempo1 tiempo2 tiempo3 tiempo4 tiempo5 tiempo6 tiempo7 tiempo8 tiempo9
tiempo10 tiempo11 tiempo12];

set(handles.perfil,'HandleVisibility','ON')
axes(handles.perfil)


a = tiempo1+tiempo2;
b = tiempo1+tiempo2+tiempo3;
c = tiempo1+tiempo2+tiempo3+tiempo4;
d = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5;
e = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6;
f = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7;
g = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8;
h = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9;
i = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10;
j =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1;
k =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1+tiempo12;



eje_tiempo=[linspace(0,tiempo1,100) linspace(tiempo1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];


eje_pot=[pot1*ones(1,100) pot2*ones(1,100) pot3*ones(1,100) pot4*ones(1,100)
pot5*ones(1,100) pot6*ones(1,100) pot7*ones(1,100) pot8*ones(1,100) pot9*ones(1,100)
pot10*ones(1,100) pot11*ones(1,100) pot12*ones(1,100)];


plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)');grid

set(handles.perfil,'HandleVisibility','OFF')


%****************************
%**** segunda figura ********
%****************************

global pot_1 tiempo_1 pot_2 tiempo_2 pot_3 tiempo_3 pot_4 tiempo_4 pot_5 tiempo_5 pot_6
tiempo_6 pot_7 tiempo_7 pot_8 tiempo_8
global pot_9 tiempo_9 pot_10 tiempo_10 pot_11 tiempo_11 pot_12 tiempo_12

pot_1=str2double(get(handles.p1_2,'String'));
tiempo_1=str2double(get(handles.t1_2,'String'));

pot_2=str2double(get(handles.p2_2,'String'));
tiempo_2=str2double(get(handles.t2_2,'String'));

pot_3=str2double(get(handles.p3_2,'String'));
tiempo_3=str2double(get(handles.t3_2,'String'));

pot_4=str2double(get(handles.p4_2,'String'));
tiempo_4=str2double(get(handles.t4_2,'String'));

pot_5=str2double(get(handles.p5_2,'String'));
tiempo_5=str2double(get(handles.t5_2,'String'));

pot_6=str2double(get(handles.p6_2,'String'));
tiempo_6=str2double(get(handles.t6_2,'String'));

pot_7=str2double(get(handles.p7_2,'String'));
tiempo_7=str2double(get(handles.t7_2,'String'));

pot_8=str2double(get(handles.p8_2,'String'));
tiempo_8=str2double(get(handles.t8_2,'String'));

pot_9=str2double(get(handles.p9_2,'String'));
tiempo_9=str2double(get(handles.t9_2,'String'));
```

```
pot_10=str2double(get(handles.p10_2,'String'));
tiempo_10=str2double(get(handles.t10_2,'String'));

pot_11=str2double(get(handles.p11_2,'String'));
tiempo_11=str2double(get(handles.t11_2,'String'));

pot_12=str2double(get(handles.p12_2,'String'));
tiempo_12=str2double(get(handles.t12_2,'String'));

potencias_1=[pot_1 pot_2 pot_3 pot_4 pot_5 pot_6 pot_7 pot_8 pot_9 pot_10 pot_11
pot_12];
tiempos_1=[tiempo_1 tiempo_2 tiempo_3 tiempo_4 tiempo_5 tiempo_6 tiempo_7 tiempo_8
tiempo_9 tiempo_10 tiempo_11 tiempo_12];

set(handles.perfil_1,'HandleVisibility','ON')
axes(handles.perfil_1)


a = tiempo_1+tiempo_2;
b = tiempo_1+tiempo_2+tiempo_3;
c = tiempo_1+tiempo_2+tiempo_3+tiempo_4;
d = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5;
e = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6;
f = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7;
g = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8;
h = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9;
i =
tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9+tiempo_
10;
j =
tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9+tiempo_
10+tiempo_11;
k =
tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9+tiempo_
10+tiempo_11+tiempo_12;



eje_tiempo=[linspace(0,tiempo_1,100) linspace(tiempo_1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];


eje_pot=[pot_1*ones(1,100) pot_2*ones(1,100) pot_3*ones(1,100) pot_4*ones(1,100)
pot_5*ones(1,100) pot_6*ones(1,100) pot_7*ones(1,100) pot_8*ones(1,100)
pot_9*ones(1,100) pot_10*ones(1,100) pot_11*ones(1,100) pot_12*ones(1,100)];


plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)');grid

set(handles.perfil_1,'HandleVisibility','OFF')



% Se generan los vectores fuente_1 y fuente_2 para la transmisiÛn

% fuente_1

fuente_1 = (ones(1,tiempos(1))* potencias(1))

for i=2:length(potencias)

    fuente_1 = cat (2,fuente_1,(ones(1,tiempos(i))* potencias(i)))

end

% fuente_2

fuente_2 = (ones(1,tiempos_1(1))* potencias_1(1))

for i=2:length(potencias_1)

    fuente_2 = cat (2,fuente_2,(ones(1,tiempos_1(i))* potencias_1(i)))
```

```
end


% Si fuente_1 y fuente_2 tienen distinto tamaÒo se igualan rellen·ndolos de ceros

vector_fuente_1=length(fuente_1);
vector_fuente_2=length(fuente_2);

if length(fuente_1)>length(fuente_2)

    x= vector_fuente_1- vector_fuente_2;
    x=zeros(1,x);
    fuente_2 = cat (2,fuente_2,x);
else
    x= vector_fuente_2- vector_fuente_1;
    x=zeros(1,x);
    fuente_1 = cat (2,fuente_1,x);
end




% ******************************
% ********* TRANSMISI"N ********
% ******************************



global serie serie_2 ans2
% Se establece la condiciÛn de parada NO
ans2 = 'No';

fopen(serie)
set(serie,'RequestToSend','on');
set(serie,'DataTerminalReady','off');
set(serie,'RequestToSend','off');
set(serie,'RequestToSend','on');
set(serie,'BaudRate',2400);
set(serie,'Terminator','CR')

fopen(serie_2)
set(serie_2,'RequestToSend','on');
set(serie_2,'DataTerminalReady','off');
set(serie_2,'RequestToSend','off');
set(serie_2,'RequestToSend','on');
set(serie_2,'BaudRate',2400);
set(serie_2,'Terminator','CR')

for i=1:max(length(fuente_1) ,length(fuente_2))
    str1=[128 37 fuente_1(i)/10];
    str2=[128 37 fuente_2(i)/10];

    chk=mod(sum(str1),256);
    chk=mod(sum(str2),256);

    str1=[str1 chk];
    str2=[str1 chk];


    % Se comprueba en cada iteraciÛn la condiciÛn de parada
    if strcmp(ans2,'Si')
        return;
    end

    % Se transmiten los valores de los vectores fuente_1 y fuente_2


    % FUENTE_1
    tic
    fwrite(serie,str1);
    fread(serie,3)
    fwrite(serie,[64 24 0 88])

    % Se comprueba el estado de alarma de la fuente
    alarma=fread(serie,3);
    alarma=alarma(2);
    alarma=dec2bin(alarma);
```

```
    ind=find(alarma=='0');
    alarma=max(ind)-1;
    switch alarma
        case '0'
            warndlg('Fallo, control del Filamento puerto 1','GEM');
        case '1'
            warndlg('Fallo, control de Potencia AnÛdica puerto 1','GEM');
        case '2'
            warndlg('Fallo, control de Temperatura puerto 1','GEM');
        case '3'
            warndlg('N.C puerto 1','GEM');
        case '4'
            warndlg('RegulaciÛn de Temperatura en progreso puerto 1','GEM');
        case '5'
            warndlg('Fallo, control PFC puerto 1','GEM');
        case '6'
            warndlg('N.C puerto 1','GEM');
        case '7'
            warndlg('Unidad deshabilitada debido a una alarma puerto 1','GEM');

    end


    pause(0.94)
    toc


    % FUENTE_2
    tic
    fwrite(serie_2,str2);
    fread(serie_2,3)
    fwrite(serie_2,[64 24 0 88])

    % Se comprueba el estado de alarma de la fuente
    alarma=fread(serie_2,3);
    alarma=alarma(2);
    alarma=dec2bin(alarma);
    ind=find(alarma=='0');
    alarma=max(ind)-1;
    switch alarma
        case '0'
            warndlg('Fallo, control del Filamento puerto 2','GEM');
        case '1'
            warndlg('Fallo, control de Potencia AnÛdica puerto 2','GEM');
        case '2'
            warndlg('Fallo, control de Temperatura puerto 2','GEM');
        case '3'
            warndlg('N.C puerto 2','GEM');
        case '4'
            warndlg('RegulaciÛn de Temperatura en progreso puerto 2','GEM');
        case '5'
            warndlg('Fallo, control PFC puerto 1','GEM');
        case '6'
            warndlg('N.C puerto 2','GEM');
        case '7'
            warndlg('Unidad deshabilitada debido a una alarma puerto 2','GEM');

    end


    pause(0.94)
    toc



end

% Se envÌa  SIEMPRE un comando de apagado de la fuente despuÈs de la
% transmisiÛn y se cierra la comunicaciÛn
str1=[128 37 0];
chk=sum(str1);
str1=[str1 chk];
fwrite(serie,str1);
fread(serie,3)
pause(0.94)
```

```
fclose(serie)


str2=[128 37 0];
chk=sum(str2);
str2=[str2 chk];
fwrite(serie_2,str2);
fread(serie_2,3)
pause(0.94)

fclose(serie_2)




% ******************
% ***** PARADA ******
% ******************

% --- Executes on button press in Parada.
function Parada_Callback(hObject, eventdata, handles)
% hObject    handle to Parada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Parada

global ans2

ans2=questdlg('øDesea detener la transmisiÛn?','GEM','Si','No','No');

if strcmp(ans2,'No')
return;
end

if strcmp(ans2,'Si')

global serie

    % Se envÌa el comando de parada
    str1=[128 37 0];
    chk=sum(str1);
    str1=[str1 chk];
        tic
        fwrite(serie,str1);
        fread(serie,3)
        pause(0.94)
        toc
fclose(serie)

return;
end

if strcmp(ans2,'Si')

global serie_2

    % Se envÌa el comando de parada
    str1=[128 37 0];
    chk=sum(str1);
    str1=[str1 chk];
        tic
        fwrite(serie_2,str1);
        fread(serie_2,3)
        pause(0.94)
        toc
fclose(serie_2)

return;
end

clearall,clc


% ******************************
```

```
% ***** PERFIL DE POTENCIA ******
% ******************************


% --- Executes on button press in Figura_Perfil_Potencia.
function Figura_Perfil_Potencia_Callback(hObject, eventdata, handles)
% hObject    handle to Figura_Perfil_Potencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Figura_Perfil_Potencia
global pot1 tiempo1 pot2 tiempo2 pot3 tiempo3 pot4 tiempo4 pot5 tiempo5 pot6 tiempo6
pot7 tiempo7 pot8 tiempo8 pot9 tiempo9 pot10 tiempo10 pot11 tiempo11 pot12 tiempo12

pot1=str2double(get(handles.p1,'String'));
tiempo1=str2double(get(handles.t1,'String'));

pot2=str2double(get(handles.p2,'String'));
tiempo2=str2double(get(handles.t2,'String'));

pot3=str2double(get(handles.p3,'String'));
tiempo3=str2double(get(handles.t3,'String'));

pot4=str2double(get(handles.p4,'String'));
tiempo4=str2double(get(handles.t4,'String'));

pot5=str2double(get(handles.p5,'String'));
tiempo5=str2double(get(handles.t5,'String'));

pot6=str2double(get(handles.p6,'String'));
tiempo6=str2double(get(handles.t6,'String'));

pot7=str2double(get(handles.p7,'String'));
tiempo7=str2double(get(handles.t7,'String'));

pot8=str2double(get(handles.p8,'String'));
tiempo8=str2double(get(handles.t8,'String'));

pot9=str2double(get(handles.p9,'String'));
tiempo9=str2double(get(handles.t9,'String'));

pot10=str2double(get(handles.p10,'String'));
tiempo10=str2double(get(handles.t10,'String'));

pot11=str2double(get(handles.p11,'String'));
tiempo11=str2double(get(handles.t11,'String'));

pot12=str2double(get(handles.p12,'String'));
tiempo12=str2double(get(handles.t12,'String'));

set(handles.perfil,'HandleVisibility','ON')
axes(handles.perfil)


a = tiempo1+tiempo2;
b = tiempo1+tiempo2+tiempo3;
c = tiempo1+tiempo2+tiempo3+tiempo4;
d = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5;
e = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6;
f = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7;
g = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8;
h = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9;
i = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10;
j =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1;
k =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1+tiempo12;



eje_tiempo=[linspace(0,tiempo1,100) linspace(tiempo1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];
```

```
eje_pot=[pot1*ones(1,100) pot2*ones(1,100) pot3*ones(1,100) pot4*ones(1,100)
pot5*ones(1,100) pot6*ones(1,100) pot7*ones(1,100) pot8*ones(1,100) pot9*ones(1,100)
pot10*ones(1,100) pot11*ones(1,100) pot12*ones(1,100)];


plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)');grid

set(handles.perfil,'HandleVisibility','OFF')

%****************************
%**** segunda figura ********
%****************************

global pot_1 tiempo_1 pot_2 tiempo_2 pot_3 tiempo_3 pot_4 tiempo_4 pot_5 tiempo_5 pot_6
tiempo_6 pot_7 tiempo_7 pot_8 tiempo_8
global pot_9 tiempo_9 pot_10 tiempo_10 pot_11 tiempo_11 pot_12 tiempo_12

pot_1=str2double(get(handles.p1_2,'String'));
tiempo_1=str2double(get(handles.t1_2,'String'));

pot_2=str2double(get(handles.p2_2,'String'));
tiempo_2=str2double(get(handles.t2_2,'String'));

pot_3=str2double(get(handles.p3_2,'String'));
tiempo_3=str2double(get(handles.t3_2,'String'));

pot_4=str2double(get(handles.p4_2,'String'));
tiempo_4=str2double(get(handles.t4_2,'String'));

pot_5=str2double(get(handles.p5_2,'String'));
tiempo_5=str2double(get(handles.t5_2,'String'));

pot_6=str2double(get(handles.p6_2,'String'));
tiempo_6=str2double(get(handles.t6_2,'String'));

pot_7=str2double(get(handles.p7_2,'String'));
tiempo_7=str2double(get(handles.t7_2,'String'));

pot_8=str2double(get(handles.p8_2,'String'));
tiempo_8=str2double(get(handles.t8_2,'String'));

pot_9=str2double(get(handles.p9_2,'String'));
tiempo_9=str2double(get(handles.t9_2,'String'));

pot_10=str2double(get(handles.p10_2,'String'));
tiempo_10=str2double(get(handles.t10_2,'String'));

pot_11=str2double(get(handles.p11_2,'String'));
tiempo_11=str2double(get(handles.t11_2,'String'));

pot_12=str2double(get(handles.p12_2,'String'));
tiempo_12=str2double(get(handles.t12_2,'String'));

potencias_1=[pot_1 pot_2 pot_3 pot_4 pot_5 pot_6 pot_7 pot_8 pot_9 pot_10 pot_11
pot_12];
tiempos_1=[tiempo_1 tiempo_2 tiempo_3 tiempo_4 tiempo_5 tiempo_6 tiempo_7 tiempo_8
tiempo_9 tiempo_10 tiempo_11 tiempo_12];

set(handles.perfil_1,'HandleVisibility','ON')
axes(handles.perfil_1)


a = tiempo_1+tiempo_2;
b = tiempo_1+tiempo_2+tiempo_3;
c = tiempo_1+tiempo_2+tiempo_3+tiempo_4;
d = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5;
e = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6;
f = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7;
g = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8;
h = tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9;
i =
tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9+tiempo_
10;
j =
tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9+tiempo_
10+tiempo_11;
```

```
k =
tiempo_1+tiempo_2+tiempo_3+tiempo_4+tiempo_5+tiempo_6+tiempo_7+tiempo_8+tiempo_9+tiempo_
10+tiempo_11+tiempo_12;


eje_tiempo=[linspace(0,tiempo_1,100) linspace(tiempo_1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];


eje_pot=[pot_1*ones(1,100) pot_2*ones(1,100) pot_3*ones(1,100) pot_4*ones(1,100)
pot_5*ones(1,100) pot_6*ones(1,100) pot_7*ones(1,100) pot_8*ones(1,100)
pot_9*ones(1,100) pot_10*ones(1,100) pot_11*ones(1,100) pot_12*ones(1,100)];


plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)');grid

set(handles.perfil_1,'HandleVisibility','OFF')
```

```
function edit26_Callback(hObject, eventdata, handles)
% hObject    handle to Temp_result (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Temp_result as text
%        str2double(get(hObject,'String')) returns contents of Temp_result as a double


% --- Executes on button press in togglebutton5.
function togglebutton5_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton5


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Se introduce el fondo del programa

% --- Executes during object creation, after setting all properties.
function upct_CreateFcn(hObject, eventdata, handles)
% hObject    handle to upct (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
axes(hObject)
imshow('Fondo.jpg')
% Hint: place code in OpeningFcn to populate upct


% ******************
% ***** Salida *****
% ******************


% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

ans=questdlg('øDesea salir del programa?','GEM','Si','No','No');
if strcmp(ans,'No')
return;
end
clear,clc,close all



% --- Executes on button press in GEM.
function GEM_Callback(hObject, eventdata, handles)
% hObject    handle to GEM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

msgbox('Grupo de Electromagnetisto y materia (UPCT)',' GEM ');


% --- Executes on button press in guardar.
function guardar_Callback(hObject, eventdata, handles)
% hObject    handle to guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- FUNCI”N DEL BOT”N "GUARDAR"

% formatos = {'*.jpg','JPEG (*.jpg)';'*.tif','TIFF (*.tif)'};
% [nomb,ruta] = uiputfile(formatos,'GUARDAR HISTOGRAMA');
% if nomb==0, return, end
% % % Crear nueva figura
%
% figura = figure;
% %Unidades y posiciÛn
% unidades  = get(handles.perfil,'Units');
% posicion   = get(handles.perfil,'Position');
% objeto_2  = copyobj(handles.perfil,figura);
%
% a=get(handles.perfil)
% %Ajustar la nueva figura
%
% set(figura,'Units',unidades);
% set(figura,'Position',[15 5 posicion(1)+60 posicion(2)+24]);
%
% % Guardar la gr·fica
% saveas(figura,[ruta nomb])
% % Cerrar figura
% close(figura)

% Crear nueva figura
figura = figure;

% Unidades y posiciÛn
unidades  = get(handles.perfil,'Units');
posicion   = get(handles.perfil,'Position');
fig  = copyobj(handles.perfil,figura);

set(figura,'Units',unidades);
set(figura,'Position',[10 5 posicion(3)+12 posicion(4)+5])


% --- Executes on button press in guardar_1.
function guardar_1_Callback(hObject, eventdata, handles)
% hObject    handle to guardar_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

figura = figure;

% Unidades y posiciÛn
unidades  = get(handles.perfil_1,'Units');
posicion   = get(handles.perfil_1,'Position');
fig  = copyobj(handles.perfil_1,figura);

set(figura,'Units',unidades);
set(figura,'Position',[10 5 posicion(3)+12 posicion(4)+5])
```

```
% --- Executes during object creation, after setting all properties.
function perfil_CreateFcn(hObject, eventdata, handles)
% hObject    handle to perfil (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate perfil



% --- Executes during object creation, after setting all properties.
function perfil_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to perfil_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate perfil_1



% --- Executes during object creation, after setting all properties.
function Inicio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called



%**************************************************
%***** SELECCI"N DE PUERTOS DE COMUNICACI"N *********
%**************************************************



function ports_Callback(hObject, eventdata, handles)
% hObject    handle to ports (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns ports contents as cell array
%        contents{get(hObject,'Value')} returns selected item from ports

global serie
puertos=instrhwinfo('serial');
puertos=puertos.AvailableSerialPorts;
puerto_serie=get(handles.ports, 'Value');
% Crea puerto serie para comunicacion
serie=serial(puertos{puerto_serie-1});
set(serie,'RequestToSend','on');
set(serie,'DataTerminalReady','off');
set(serie,'BaudRate',2400);
set(serie,'Terminator','CR')


% --- Executes during object creation, after setting all properties.
function ports_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ports (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% ************************************************************
% Para transmitir por el otro puerto cambiamos serie por serie_2
% ************************************************************
```

```
% --- Executes on selection change in ports_1.
function ports_1_Callback(hObject, eventdata, handles)
% hObject    handle to ports_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns ports_1 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from ports_1

global serie_2
puertos=instrhwinfo('serial');
puertos=puertos.AvailableSerialPorts;
puerto_serie=get(handles.ports, 'Value');
% Crea puerto serie para comunicacion
serie_2=serial(puertos{puerto_serie-1});
set(serie_2,'RequestToSend','on');
set(serie_2,'DataTerminalReady','off');
set(serie_2,'BaudRate',2400);
set(serie_2,'Terminator','CR')

% --- Executes during object creation, after setting all properties.
function ports_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ports_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

8.3 Código Interfaz Funciones.

```
function varargout = Interfaz_funciones(varargin)

%
% Universidad PolitÈcnica De Cartagena (upct)
% Grupo de investigaciÛn GEM (Grupo de Electromagnetismo y Materia)
%
% Proyecto Final de Carrera Ing.Superior de TelecomunicaciÛn
% TÌtulo: RealizaciÛn De Un Programa Con Interfaz De Usuario Para La
% GeneraciÛn De Patrones De Potencia Arbitratios Con La Fuente De AlimentaciÛn Magdrive
%
% Alumno: Pablo Berm·dez Alem·n
% Director de Proyecto: Ju·n MonzÛ Cabrera
% Codirector: Francisco Javier Clemente Fern·ndez
%
% INTERFAZ_FUNCIONES M-file for Interfaz_funciones.fig
%      INTERFAZ_FUNCIONES, by itself, creates a new INTERFAZ_FUNCIONES or raises the
existing
%      singleton*.
%
%      H = INTERFAZ_FUNCIONES returns the handle to a new INTERFAZ_FUNCIONES or the
handle to
%      the existing singleton*.
%
%      INTERFAZ_FUNCIONES('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFAZ_FUNCIONES.M with the given input arguments.
%
%      INTERFAZ_FUNCIONES('Property','Value',...) creates a new INTERFAZ_FUNCIONES or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Interfaz_funciones_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Interfaz_funciones_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Interfaz_funciones

% Last Modified by GUIDE v2.5 22-Nov-2011 00:50:43

% Begin initialization code - DO NOT EDIT
clc

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_funciones_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_funciones_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


% End initialization code - DO NOT EDIT


% --- Executes just before Interfaz_funciones is made visible.
function Interfaz_funciones_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interfaz_funciones (see VARARGIN)
```

```
% Choose default command line output for Interfaz_funciones
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


%Coloca una imagen en cada botÛn
[a,map]=imread('logogem.jpg');
[r,c,d]=size(a);
x=ceil(r/50);
y=ceil(c/60);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.GEM,'CData',g);



% Crea puerto serie para comunicacion
global serie
serie=serial('COM1');
set(serie,'RequestToSend','on');
set(serie,'DataTerminalReady','off');
set(serie,'BaudRate',2400);
set(serie,'Terminator','CR')

%captura

handles.output = hObject;
% Update handles structure
handles.rgb = [];



% UIWAIT makes Interfaz_funciones wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_funciones_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% Condiciones para introducir los datos de Tiempos y Potencias


% ****************************
% ****Condiciones Potencias****
% ****************************


function varargout = p1_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p1,'string'));
if isnan(c1)

    c1=0;

    set(handles.p1,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p1,'String', c1);
```

```
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p1,'String', c1);

end

function varargout = p2_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p2,'string'));
if isnan(c1)

    c1=0;

    set(handles.p2,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p2,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p2,'String', c1);

end

function varargout = p3_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p3,'string'));
if isnan(c1)

    c1=0;

    set(handles.p3,'String', c1);
```

```
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p3,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p3,'String', c1);

end

function varargout = p4_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p4,'string'));
if isnan(c1)

    c1=0;

    set(handles.p4,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p4,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

   if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p4,'String', c1);

end

function varargout = p5_Callback(h, eventdata, handles, varargin)
```

```
Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p5,'string'));
if isnan(c1)

    c1=0;

    set(handles.p5,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p5,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p5,'String', c1);

end

function varargout = p6_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p6,'string'));
if isnan(c1)

    c1=0;

    set(handles.p6,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p6,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
```

```
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p6,'String', c1);

end

function varargout = p7_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p7,'string'));
if isnan(c1)

    c1=0;

    set(handles.p7,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p7,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p7,'String', c1);

end

function varargout = p8_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p8,'string'));
if isnan(c1)

    c1=0;

    set(handles.p8,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p8,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;
```

```
    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p8,'String', c1);

end

function varargout = p9_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p9,'string'));
if isnan(c1)

    c1=0;

    set(handles.p9,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p9,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

    if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p9,'String', c1);

end

function varargout = p10_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p10,'string'));
if isnan(c1)

    c1=0;

    set(handles.p10,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;
```

```
    set(handles.p10,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

   if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p10,'String', c1);

end

function varargout = p11_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p11,'string'));
if isnan(c1)

    c1=0;

    set(handles.p11,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p11,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

  if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p11,'String', c1);

end

function varargout = p12_Callback(h, eventdata, handles, varargin)

Potencias = [80 90 100 200 300 400 500 800 1000 1300];

c1 = str2double(get(handles.p12,'string'));
if isnan(c1)

    c1=0;
```

```
    set(handles.p12,'String', c1);

errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0

    c1=0;

    set(handles.p12,'String', c1);

errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
if c1 ~= Potencias

    c1=c1/10;
    c1=round(c1);
    c1=c1*10;

  if c1 > 1300
        c1=1300;
        errordlg('El valor como m·ximo introducido debe ser 1300 W ', 'Bad Input',
'modal')
    end

    if c1<80
        c1=80;
        errordlg('El valor como mÌnimo introducido debe ser 80 W ', 'Bad Input',
'modal')
    end

    set(handles.p12,'String', c1);

end


% ******************
% Condiciones Tiempos
% ******************


function varargout = t1_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t1,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t2_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t2,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t3_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t3,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t4_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t4,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end
```

```
function varargout = t5_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t5,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t6_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t6,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t7_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t7,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t8_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t8,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t9_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t9,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t10_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t10,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t11_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t11,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end

function varargout = t12_Callback(h, eventdata, handles, varargin)
c1 = str2double(get(handles.t12,'string'));
if isnan(c1)
errordlg('El valor introducido debe ser numÈrico', 'Bad Input', 'modal')
end
if c1<0
errordlg('El valor introducido debe ser positivo ', 'Bad Input', 'modal')
end



% --- Executes during object creation, after setting all properties.
function p1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p1 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function p5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function p12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function t9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function t10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function t11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes during object creation, after setting all properties.
function t12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to t12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Inicio.
function Inicio_Callback(hObject, eventdata, handles)
% hObject    handle to Inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Inicio
global pot1 tiempo1 pot2 tiempo2 pot3 tiempo3 pot4 tiempo4 pot5 tiempo5 pot6 tiempo6
pot7 tiempo7 pot8 tiempo8
global pot9 tiempo9 pot10 tiempo10 pot11 tiempo11 pot12 tiempo12

pot1=str2double(get(handles.p1,'String'));
tiempo1=str2double(get(handles.t1,'String'));

pot2=str2double(get(handles.p2,'String'));
```

```
tiempo2=str2double(get(handles.t2,'String'));

pot3=str2double(get(handles.p3,'String'));
tiempo3=str2double(get(handles.t3,'String'));

pot4=str2double(get(handles.p4,'String'));
tiempo4=str2double(get(handles.t4,'String'));

pot5=str2double(get(handles.p5,'String'));
tiempo5=str2double(get(handles.t5,'String'));

pot6=str2double(get(handles.p6,'String'));
tiempo6=str2double(get(handles.t6,'String'));

pot7=str2double(get(handles.p7,'String'));
tiempo7=str2double(get(handles.t7,'String'));

pot8=str2double(get(handles.p8,'String'));
tiempo8=str2double(get(handles.t8,'String'));

pot9=str2double(get(handles.p9,'String'));
tiempo9=str2double(get(handles.t9,'String'));

pot10=str2double(get(handles.p10,'String'));
tiempo10=str2double(get(handles.t10,'String'));

pot11=str2double(get(handles.p11,'String'));
tiempo11=str2double(get(handles.t11,'String'));

pot12=str2double(get(handles.p12,'String'));
tiempo12=str2double(get(handles.t12,'String'));

potencias=[pot1 pot2 pot3 pot4 pot5 pot6 pot7 pot8 pot9 pot10 pot11 pot12];
tiempos=[tiempo1 tiempo2 tiempo3 tiempo4 tiempo5 tiempo6 tiempo7 tiempo8 tiempo9
tiempo10 tiempo11 tiempo12];



[valores,indices]=find(potencias)
for i=1:max(indices)
    str1=[128 37 potencias(i)/10]
end
%
% Se crean los vectores de valores y potencias para ver dÛnde se han
% introducido las potencias
%
% [valores,indices]=find(potencias);
% global serie
% fopen(serie)
% set(serie,'RequestToSend','on');
% set(serie,'DataTerminalReady','off');
% set(serie,'RequestToSend','off');
% set(serie,'RequestToSend','on');
% set(serie,'BaudRate',2400);
% set(serie,'Terminator','CR')
% for i=1:max(indices)
%     str1=[128 37 potencias(i)/10];
%     chk=sum(str1);
%     str1=[str1 chk];
%     for j=1:tiempos(i)
%         tic
%         fwrite(serie,str1);
%         fread(serie,3)
%         pause(0.94)
%         toc
%     end
% end
% fclose(serie)
%
%
%
%
%
% % numerocomandos=tiempo
%
```

```
% --- Executes on button press in Parada.
function Parada_Callback(hObject, eventdata, handles)
% hObject    handle to Parada (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Parada

ans=questdlg('øDesea detener la transmisiÛn?','GEM','Si','No','No');
if strcmp(ans,'No')
return;
end


if strcmp(ans,'Si')
% Cuando se pulsa parada la fuente deja de transmitir
%
% [valores,indices]=find(potencias);
% global serie
% fopen(serie)
% set(serie,'RequestToSend','on');
% set(serie,'DataTerminalReady','off');
% set(serie,'RequestToSend','off');
% set(serie,'RequestToSend','on');
% set(serie,'BaudRate',2400);
% set(serie,'Terminator','CR')
% for i=1:max(indices)
%
% Cuando se transmite el tercer bit a 0 se para
%
%     str1=[128 37 0];
%     chk=sum(str1);
%     str1=[str1 chk];
%     for j=1:tiempos(i)
%         tic
%         fwrite(serie,str1);
%         fread(serie,3)
%         pause(0.94)
%         toc
%     end
% end
% fclose(serie)

return;
end

clearall,clc


% --- Executes on button press in Figura_Perfil_Potencia.
function Figura_Perfil_Potencia_Callback(hObject, eventdata, handles)
% hObject    handle to Figura_Perfil_Potencia (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Figura_Perfil_Potencia
global pot1 tiempo1 pot2 tiempo2 pot3 tiempo3 pot4 tiempo4 pot5 tiempo5 pot6 tiempo6
pot7 tiempo7 pot8 tiempo8 pot9 tiempo9 pot10 tiempo10 pot11 tiempo11 pot12 tiempo12


set(handles.perfil,'HandleVisibility','ON')
axes(handles.perfil)


a = tiempo1+tiempo2;
b = tiempo1+tiempo2+tiempo3;
c = tiempo1+tiempo2+tiempo3+tiempo4;
d = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5;
e = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6;
f = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7;
g = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8;
h = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9;
i = tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10;
```

```
j =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1;
k =
tiempo1+tiempo2+tiempo3+tiempo4+tiempo5+tiempo6+tiempo7+tiempo8+tiempo9+tiempo10+tiempo1
1+tiempo12;



eje_tiempo=[linspace(0,tiempo1,100) linspace(tiempo1,a,100) linspace(a,b,100)
linspace(b,c,100) linspace(c,d,100) linspace(d,e,100) linspace(e,f,100)
linspace(f,g,100) linspace(g,h,100) linspace(h,i,100) linspace(i,j,100)
linspace(j,k,100)];


eje_pot=[pot1*ones(1,100) pot2*ones(1,100) pot3*ones(1,100) pot4*ones(1,100)
pot5*ones(1,100) pot6*ones(1,100) pot7*ones(1,100) pot8*ones(1,100) pot9*ones(1,100)
pot10*ones(1,100) pot11*ones(1,100) pot12*ones(1,100)];


 plot(eje_tiempo,eje_pot);xlabel('Tiempo (s)'),ylabel('Potencia(W)')
%(handles.perfil,'HandleVisibility','OFF')
```

```
function edit26_Callback(hObject, eventdata, handles)
% hObject    handle to Temp_result (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Temp_result as text
%        str2double(get(hObject,'String')) returns contents of Temp_result as a double


% --- Executes during object creation, after setting all properties.
function Temp_result_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Temp_result (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in togglebutton5.
function togglebutton5_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton5


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% Se introduce el fondo del programa

% --- Executes during object creation, after setting all properties.
function upct_CreateFcn(hObject, eventdata, handles)
% hObject    handle to upct (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
axes(hObject)
imshow('Fondo.jpg')
% Hint: place code in OpeningFcn to populate upct


% --- Executes on button press in Temperatura.
function Temperatura_Callback(hObject, eventdata, handles)
% hObject    handle to Temperatura (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% como podemos meter el valor medido de la temperatura?

% Resultado =  s3 o s4 ( la temperatura de control fuente ) num2str ()

% set(handles.Temp_result,'String',Resultado);



% ******************
% ***** Salida *****
% ******************


% --- Executes on button press in salir.
function salir_Callback(hObject, eventdata, handles)
% hObject    handle to salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

ans=questdlg('øDesea salir del programa?','GEM','Si','No','No');
if strcmp(ans,'No')
return;
end
clear,clc,close all



% --- Executes on button press in GEM.
function GEM_Callback(hObject, eventdata, handles)
% hObject    handle to GEM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

msgbox('Grupo de Electromagnetisto y materia (UPCT)',' GEM ');


% --- Executes on button press in guardar.
function guardar_Callback(hObject, eventdata, handles)
% hObject    handle to guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- FUNCI”N DEL BOT”N "GUARDAR"


% Crear nueva figura
figura = figure;

% Unidades y posiciÛn
unidades  = get(handles.perfil,'Units');
posicion   = get(handles.perfil,'Position');
fig  = copyobj(handles.perfil,figura);

% Modificar la nueva figura
set(fig,'Units',unidades);
set(fig,'Position',[15 5 posicion(3) posicion(4)]);



% ***************************
% Guardar de forma autom·tica
% ***************************

% % Guardar archivo
% formatos = {'*.jpg','JPEG (*.jpg)';'*.tif','TIFF (*.tif)';'*.bmp','BMP (*.bmp)'};
```

```
% [nomb,ruta] = uiputfile(formatos,'GUARDAR PERFIL DE POTENCIA');
% if nomb==0, return, end
%
% % Crear nueva figura
%  figura = figure;
%
% % Unidades y posiciÛn
% unidades  = get(handles.perfil,'Units');
% posicion   = get(handles.perfil,'Position');
% objeto_2  = copyobj(handles.perfil,figura);
% % Modificar la nueva figura
% set(objeto_2,'Units',unidades);
% set(objeto_2,'Position',[15 5 posicion(3) posicion(4)]);
% % Ajustar la nueva figura
% set(figura,'Units',unidades);
% set(figura,'Position',[10 5 posicion(3)+30 posicion(4)+10]);
% %Guardar la gr·fica
% saveas(figura,[ruta nomb])
% %Cerrar figura
% close(figura)



% --- Executes during object creation, after setting all properties.
function perfil_CreateFcn(hObject, eventdata, handles)
% hObject    handle to perfil (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate perfil


% --- Executes during object creation, after setting all properties.
function Inicio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Inicio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

9. Agradecimientos.

La realización de este proyecto es más que el último paso para terminar la carrera. Significa el final de una de las etapas más felices de mi vida para comenzar una nueva, la cual espero que pueda hacerme tan feliz como lo ha hecho ésta.

No habría llegado a la meta de este largo recorrido sin la ayuda de aquellos que me han acompañado durante estos años, a quienes me gustaría dedicar este proyecto.

En primer lugar, me gustaría darle las gracias a Juan Monzó por confiar en mi desde el primer momento y darme la oportunidad de poder realizar este proyecto. A Francisco Clemente, por tu generosidad, tu disponibilidad y por toda la ayuda que me has ofrecido. Este proyecto no habría sido posible sin vosotros.

Gracias a todos mis profesores por enseñarme los conocimientos necesarios para llegar a ser ingeniero, pero sobretodo, gracias por lecciones de vida que habéis transmitido, que son las que nos harán marcar diferencia a los alumnos una vez salgamos de la universidad, ya que sé por experiencia propia que este trabajo extra que realizáis no se hace en ninguna otra universidad. En especial a José Luís Gómez y José Víctor Rodríguez por haberme ayudado en momentos en los cuales he estado apunto de tirar la toalla haciéndome ver las cosas desde otro punto de vista. Mención especial a Ambrosio (el psicólogo de Teleco).

A mi familia por ayudarme a ser tal y como soy, por enseñarme a quererme y respetarme a mi mismo y a los demás , por su infinita paciencia y apoyo incondicional en todo momento, por aguantarme cuando he sido insufrible, sobretodo en la época de exámenes y por enseñarme a que los obstáculos siempre los podremos superar juntos.

Gracias a mis amigos, Carica (Alias = Vidita by Víctor), sin tu amistad durante todos estos años mi vida actual no sería como es ahora. Mª Dolores, eres el mejor regalo que he recibido en todos estos años, que creo que han marcado un antes y un después en nosotros. Paco, gracias por todos los buenos momentos, en prácticas, en clase, en sesiones de estudio interminables, fiestas, delegación, etc. etc. etc. sin tu ayuda habrían sido imposibles. Rocío, espero que Paicho crezca tanto como ha crecido nuestra amistad. Lola, de una forma u otra siempre estás ahí cuando se te necesita eres única, Merce, gracias por ser como eres y cuidarme tanto desde el primer momento en que nos conocimos. También me gustaría dar las gracias a Eva, te deseo la máxima felicidad en tu nueva vida, Magalí los momentos compartidos contigo no tienen precio, Elisa por ser una de las personas más dulces que he conocido en mi vida, Almudena, Jorge, María, Antonio y Mª Ángeles también gracias infinitas a vosotros.

Gracias a mi gran familia de J.M.V que me han guiado todos estos años, pudiendo decir que en cada rincón de España tengo un amigo verdadero. (Isa, Mixi, Maria (Fresón), Sonia, Fran, Nieves, Edu … la lista es interminable).

Gracias a mis Erasmus, tanto los de Cartagena como los de Italia. Edo, grazie per la tua amicizia, per me sei un vero amico. Rob Power avevi ragione, alla fine ho diventato un italiano. Alessandro grazie per tutto il aiuto in questo tempo a Torino. Stefano e Lely voi due siete grandi tutto andra bene. Paola, sei piccola ma grande grande grande grande di cuore, ti voglio bene. Rocío, eres el mejor recuerdo que me llevo del Erasmus a Torino. Andrea, sei il mio polentone preferito mi perra :D. Alberto grazie mille per tante cose…mi mancano le parole per te. WakaGiuseppe e Leo Bianco spero di rivederci in Africa o in Europa. Silvia, Elisa y Jorgito siete una parte molto importante per me in questo ultimo tempo. Inma ,Jesús y Alicia, grazie mille per questo tempo insieme.

A mis profes de inglés Juana Mari y Marta por pasar de ser dos profesoras a convertirse en dos grande amigas. Hayat, Pedro, Adonis y Manu, porque aunque nos hemos conocido al final de este periodo hemos creado un vínculo muy especial, que espero que crezca a lo largo del tiempo.

Gracias a todos vosotros, tanto a lo que he nombrado como los que no, porque si soy quien soy y he llegado hasta donde he llegado, en gran medida es gracias a vosotros.

STAND BY ME

Pablo